Bana István

Az SSADM

rendszerszervezési módszertan

Dr. Bana István

Az SSADM
rendszerszervezési módszertan

Nyitott rendszerű képzés - távoktatás -
oktatási segédlete * Tankönyv

Lektorálta: Pirkó József

LSI Oktatóközpont
A Mikroelektronika Alkalmazásának
Kultúrájáért Alapítvány

I S B N 963 577 129 0

Kiadó: **LSI Oktatóközpont**
Felelős vezető: **Dr. Kovács Magda**
Témafelelős: **Flier István**

LIGATURA KFT - VÁCI ÁFÉSZ NYOMDA

Tartalomjegyzék

Előszó

A Miniszterelnöki Hivatal Informatikai Tárcaközi Bizottsága úgy döntött, hogy a világpiacon létező számos strukturált rendszerszervezési módszertan közül az SSADM-et ajánlja követendő szabványként. Ezáltal jelent meg *széleskörű* érdeklődés Magyarországon az SSADM iránt, amelyet azonban már 1989 óta magam is SZÁMALK továbbképző tanfolyamokon ismertettem.

Azáltal, hogy a módszertan hivatalosan ajánlottá vált, a számítástechnikai oktatásban egyre szélesebb körben szükséges az ismertetése, ugyanakkor eddig nem állt rendelkezésre tankönyv az oktatási munka támogatásához. A KFKI Információtechnológia Alapítvány elkészítette az eredeti angol módszertani kézikönyvek tömörített és rövidített magyar változatát, amely azonban tankönyvként igen nehezen használható, hiszen eredetijének sem ez a célja.

Ezt a hiányt próbálja pótolni ez a tankönyv. Mint minden könyv esetében, itt is meg kellett találni a megfelelő egyensúlyt a megjelentetés gyorsasága, a költségek, a rendelkezésre álló idő és a minőség között. Az a kompromisszum, amelynek eredményeként ez a könyv megszületett, nem tűzte ki célul azt, hogy mindent elmondjon az SSADM-ről, amit egyáltalán el lehet. Sokkal inkább célja e könyvnek, hogy átsegítse a megismerés korai nehézségein mindazokat az olvasókat, akik rendszerszervezésben és/vagy programozásban már rendelkeznek bizonyos alapvető jártassággal.

Ne várja tehát a kedves Olvasó, hogy ebből a könyvből megtanulhatja a rendszerszervezés alapjait, és azt sem, hogy mindent tudni fog az SSADM-ről, ha ezt elolvassa. Azt viszont - reményeim szerint - tudni fogja, hogy mi az SSADM, miért jelent meg, milyen a felépítése, mire és hogyan használható.

A könyv terjedelme nem tette lehetővé, a magyar terminológia kialakulatlansága pedig nem tette célszerűvé a módszertan formanyomtatványainak közlését és magyarázatát. Ezek közül a legfontosabbak megtalálhatók az Információtechnológia Alapítványnak az Irodalomjegyzékben közölt kiadványában, amely e tankönyv anyagának ismeretében magyar nyelvű

referencia-kézikönyvként használható. Az angolul értő olvasóknak javaslom az eredeti hivatkozási kézikönyvek (ld. Irodalomjegyzék) tanulmányozását, elsősorban teljeskörűségük miatt.

Ugyancsak terjedelmi és időbeni korlátok miatt kellett eltekintenem egy átfogó esettanulmány közlésétől. A közölt példák tehát egymástól függetlenül értelmezendők, még ha sokuk azonos környezetből származik is. Kivételt képeznek persze azok az esetek, ahol kifejezett hivatkozás van más példára.

A könyv hármas tagolású: az első rész módszertani alapokkal foglalkozik, a második az SSADM szerkezetét, a harmadik pedig a technikáit ismerteti.

Az SSADM szerkezetével kapcsolatos ismertetés felfogását talán szerénytelenség nélkül lehet újszerűnek nevezni, mert a modulokat és szakaszokat a szokásossal ellenkező irányban haladva fejti ki, ami - véleményem szerint - inkább megfelel a módszertan egyik lényegét jelentő termékszemléletnek, sőt a didaktikának is. Remélem, hogy az Olvasó hasonlóan fogja érezni.

A technikák ismertetése nem teljeskörű, de valamennyi diagramokon alapuló technikát tartalmazza, ami ismét összhangban van az SSADM alapfelfogásával. Az elhagyott technikák lényegében nem SSADM-specifikusak (pl. helyzetfelmérés, rendszertechnikai változatok kialakítása, fizikai tervezés).

Köszönetnyilvánítás
Surányiné Benedikt Verának a bizalomért és ösztönzésért, Lugosi Csabának a kézirat szerkesztésében nyújtott segítségéért, Feleségemnek a felfordulás elviseléséért és Flier Istvánnak a megértésért és a kiadás irányításáért.

Budapest, 1994. augusztus

dr Bana István

ELSŐ RÉSZ

AZ INFORMÁCIÓS RENDSZEREK STRUKTURÁLT SZEMLÉLETE

1. fejezet
Bevezetés

A strukturált módszertanok - és ezen belül a Strukturált Rendszerelemzési és Tervezési Módszer (Structured Systems Analysis and Design Method) - hosszú fejlődés eredményeként alakultak ki, amely fejlődésnek pontosan követhető a menete és az okai. Ez a tankönyv természetesen nem a rendszerszervezés történetével foglalkozik, de éppen mert tankönyvről van szó, bizonyos mértékig feltétlenül érdemes visszatekintenünk az eddig megtett útra, hiszen ebből önmagában véve is sokat lehet tanulni ("Más kárán tanul az okos!").

1.1 A kezdetek

A számítógépek u.n. üzleti célú alkalmazása az 50-es években indult meg. Először a programozás technikái alakultak ki, mégpedig a hardver fejlettségi szintjének megfelelően. Ez az ötvenes években annyit jelentett, hogy a programoknak rendkívül helytakarékosaknak kellett lenniük. A központi memória mérete legfeljebb néhány Kbyte nagyságú programok futtatását tette lehetővé.

Az alkalmazott technika akkori újszerűsége, valamint korlátai együttesen eredményezték, hogy a rendszereket (eleinte inkább csak feldolgozásokat) egymástól elszigetelten tervezték.

Ebben a nagyon korai időszakban a rendszerszervezési jellegű munka célja az volt, hogy minél pontosabban meghatározza a programozó számára az *előállítandó outputokat*, amelyek döntően papír-outputok voltak és tablónak nevezték őket (akkor és még igen hosszú ideig).

Az adattárolás technikái ugyancsak meglehetősen kezdetlegesek voltak. Az első időkben a feldolgozásokhoz szükséges adatokat legnagyobb részt inputként vitték be először celluloid, majd papír adathordozón (lyukkártya, lyukszalag). Még a hosszabb ideig állandó (törzs-) adatok tárolását is így oldották meg, ami a lyukkártyás adatfeldolgozásból átvett technika volt. Később - még ugyanebben a korszakban - fejlődtek az adattárolás módszerei, megjelentek a mágneses adathordozók, a mágnesdob és a mágnesszalag, ezzel együtt pedig a háttértároló fogalma (szemben a központi tárral).

A háttértár magával hozta a címezhetőség problémakörét és innen már csak egy lépés volt a fájlszervezés, mint külön rendszerszervezési tématerület megjelenése. A mágneslemezek tömeges elterjedése különösen nagy lendületet adott a fájlszervezési technikák fejlődésének, mert ez volt az a háttértároló eszköz, amely - a széles körű alkalmazásba nem került mágnesdob mellett - a tárolóhelyek címzését lehetővé tette, ha korlátozottan is. A módszertan viszonylagos lemaradásának egyik szép példája lehet éppen a mágneslemez tárolók megjelenése. Abban a pillanatban, amikor adottá vált a széles körben használatos címezhető háttértár, elvileg már át lehetett volna térni a hagyományos fájlszervezési módszerekről az adatbázis kezelésre és ezzel együtt egészen más megvilágításba került volna az információs rendszerek integrálhatósága, valamint egy sor további problémakör. Ehhez azonban előbb fel kellett fedezni a mágneslemezben rejlő lehetőségek teljes skáláját, meg kellett alkotni az adatbáziskezelő rendszereket (mint szoftver termékeket), és ki kellett dolgozni az ilyen környezetben hatékony rendszerszervezési technikákat (pl. adatmodellezés).

Eközben maga a rendszerszervezés továbbra is úgy jellemezhető, mint *output-orientált rendszerszervezés*. Az egész munka mindig azzal indult, hogy minél pontosabban sikerüljön meghatározni a felhasználó által igényelt outputokat (tablók), ezt követően az outputok előállításához szükséges algoritmusokat, valamint az algoritmusok adatigényét. Ezen a ponton kétfelé vált a szervezési munka: el kellett dönteni, hogy az algoritmusok adatigényét milyen arányban elégítjük ki huzamosabban tárolt - vagyis u.n. törzs jellegü - adatokból, ill. esetenként, kifejezetten az adott feldolgozáshoz bevitt input adatokból, amelyeket kezdtünk *tranzakciós* adatoknak nevezni.

További fontos jellemzője ennek a korszaknak, hogy a *feldolgozásnak* van központi szerepe. Minden további rendszer-összetevőt e köré kellett csoportosítani. A feldolgozások miatt kellett az általuk szolgáltatandó outputokból kiindulni, majd a szintén általuk igényelt adatokkal foglalkozni. Ekkor tehát *az adatok még csak mint a feldolgozások szükségletei jelentek meg.*

Nagymértékben összefüggött ez a helyzet azzal a körülménnyel, hogy ekkor örülni kellett, ha korlátozott környezetben, szűk felhasználói kör számára az igényeknek megfelelő *kis* rendszereket sikerült megszervezni.

Közben persze rohamléptekkel haladt előre a hardver fejlesztése, valamint egyre inkább bevonult a köztudatba, hogy létezik a számítástechnika. Ez azt eredményezte, hogy egy-egy szervezeten (vállalaton, intézményen) belül is kezdtek szaporodni a - még egyenlőre kis - számítógépes információrendszerek. Ennek

eredményeként egyre sürgetőbben vetődött fel ezek koordinációjának kérdése. A gondok főképpen az adatokkal kapcsolatban jelentkeztek. Kiderült ugyanis, hogy az ugyanazon szervezeten (pl. vállalaton) belül addig egymástól elszigetelt módon létrehozott számítógépes rendszerek

- rengeteg adatot átfedő módon tartalmaztak, valamint

- az adatok nevei nem egységesen kerültek meghatározásra.

Az elsőként említett gond jelentkezett időben is korábban és a *redundancia* problémaköreként vonult be a szakmai köztudatba. Először úgy tűnt, hogy a legnagyobb gond az adatok ismételt (átfedő) tárolásával kapcsolatban az, hogy pazarolja az egyébként is szűkösen rendelkezésre álló háttértároló kapacitást.

Összefoglalóan tehát úgy lehet jellemezni a rendszerszervezési módszerek fejlődésének első korszakát, mint amely alapelvei tekintetében *feldolgozásközpontos* volt, technikailag pedig a legdöntőbb jellemzője az output-orientáltság volt: a kívánt végeredmény meghatározásából "pörgette vissza" az egész rendszer felépítését.

1.2 Az adatok a középpontba kerülnek

A jó programozó alapjában véve lusta emberfajta. Biztos hogy ezt - az első pillanatban talán nagyon negatívnak tűnő - megállapítást maguk az érintettek nem sérelmezik. Azért nem, mert az ő lustaságuk *alkotó lustaság*. Egyszerűen nem szeretik ismételten elvégezni az ismétlődően jelentkező feladatokat, inkább kitalálnak valamilyen általánosított megoldást, amit aztán mindig elő lehet húzni, amikor a feladat jelentkezik.

Ilyen feladat volt az adatok meghatározásának és kezelésének feladata. A programnyelvek (mint pl. COBOL) addig is tartalmazták az ehhez szükséges eszközöket, de úgy, hogy magát az adat definiciót azért minden programba külön bele kellett foglalni. Megjelent tehát az igény arra, hogy az adatkezelési feladatokat ellátó rutinokat általánosítsák.

Ez volt az a pillanat, amikor a rendszerszervezési oldalon tapasztalt gondok (redundancia, elnevezések egységesítése), a programozói törekvések a munka hatékonyságának növelésére, valamint a hardver fejlesztése (címezhető háttértároló) találkoztak és a problémák megoldására, ill. a szerteágazó rutinmunkák megkönnyítésére *megszületett az adatbáziskezelés elve*.

Eszerint lehetett (és kellett) olyan szoftver terméket előállítani, amely az adatok meghatározásának és kezelésének gondját egységesen és általánosítottan oldja meg. Ez lett az *adatbázis kezelő rendszer*. Ha ezt a folyamatot időben akarjuk elhelyezni, akkor azt mondhatjuk, hogy kb. egy évtizedet, a hatvanas éveket vette igénybe ez a törekvés. 1971 áprilisában már egy alaposan kidolgozott és szabványnak tekintett dokumentum is megjelent ezzel kapcsolatban, a híres CODASYL jelentés (a rövidítés az adatkezeléssel foglalkozó nemzetközi bizottság neve), amely a hálós modellen alapuló adatbázisok szabványosításával foglalkozott. Hozzávetőlegesen ezzel egyidőben jelent meg a relációs adatbázisok elve is.

Az adatbáziskezelő rendszerek megjelenése fokozatosan és gyökeresen átalakította a rendszerszervezéssel kapcsolatos szemléletet. Egy nagyon lényeges súlypont eltolódás következett be: az addigi output-orientáltság helyett mindinkább az *adat-orientáltság* szemlélete tört előre. Kezdtük felismerni, hogy az output nem mindenható. Újabb és újabb outputokat kitalálhat bármely pillanatban egy-egy felhasználó, ami azonban viszonylag stabilan, hosszabb távon jellemzi egy szervezet működését, az a szervezet adatbázisa. Ha ezt sikerül megfelelően - azaz valósághűen - feltárni és meghatározni, akkor hosszú távra megvethetjük az adott szervezet teljes (vagyis integrált) információs rendszerének alapját. Ezzel együtt járt annak a nézetnek az elterjedése, mely szerint nemcsak arról van szó, hogy az adatoknak kiemelt fontossága van információs rendszerek, ill. szoftvercsomagok fejlesztésekor, de maguknak a fejlesztési tevékenységeknek a sorát is az adatok felmérésével és az adatmodellezéssel kell elkezdeni.

Természetesen megjelentek túlzó és szakszerűtlen nézetek is. Talán a legtipikusabb ezek közül az, ami az u.n *redundancia* témakörében jelentkezett, és mintegy az előző korszak visszahatásaként értelmezhető.

Akkor, amikor korábban rendkívül szoros volt a kapcsolat a programok és az általuk használt adatok között (u.n. program-adat függőség), tehát a programok mintegy "kisajátították" az adatokat, rendkívül nagyfokú volt a programok igényei szerint megszervezett fájlok tartalmában az átfedés, másszóval a redundancia. Ebből azután többirányú és meglehetősen kellemetlen gondok adódtak:

1. A még mindig szűkösen rendelkezésre álló háttértároló kapacitás pocsékolása.

2. A többszörösen tárolt adatok összehangolatlansága, ami többnyire abban nyilvánult meg, hogy a különböző fájlokban ugyanazt az adatot különböző

periódussal tartották karban és ez az adatok ellentmondásos megjelenéséhez vezetett. Nem lehetett tudni, hogy egy adott pillanatban ugyanazon adat mely fájlból származó értékét lehet mérvadónak tekinteni a rendszer egészére vonatkozóan.

Figyelemre méltó, hogy míg az előző, pontban említett gond tisztán technikai jellegü, addig az utóbbi a felhasználók munkáját közvetlenül és alapvetően zavarta.

Az adatbáziskezelés megjelenése, azáltal, hogy az adatok közötti kapcsolatteremtés lehetőségét kínálta, azt a benyomást keltette, hogy ezeken a kapcsolatokon keresztül bármelyik program hozzá tud férni bármelyik adathoz, tehát a redundáns (többszörös) tárolás *abszolút módon* kiküszöbölhető.

Ez az elgondolás igen szakszerűtlennek és tévesnek bizonyult, kiderült ugyanis, hogy az adatok közötti kapcsolatteremtés biztonságos megoldásához és a kapcsolatok fenntartásához *feltétlenül* szükség van meghatározott szintű redundancia fenntartására, amely szintet *optimálisnak* tekinthetjük. Nem a redundancia minimumára, hanem az optimumára kell tehát törekedni.

Hogy ezt milyen módon lehet elérni, milyen rendszerszervezési technikákat kell alkalmazni, hogyan lehet tehát a legmegfelelőbb adatszerkezeteket meghatározni - ez volt a rendszerszervezés második nagy korszakának a központi kérdése. Időben kb. a 70-es évtizedre tehető ez a korszak.

Ekkor került kidolgozásra az *adatmodellezés* elmélete és gyakorlata. A korszak legjelentősebb adatkezelési szakemberei: *Bachman* (az adatbázis kezelés első számú úttörője), *Codd* (a relációs adatbázisok elméletének kidolgozója), *Halassy* (a fogalmi szintű adatmodellezés egységes elméletének megalkotója).

A rendszerszervezés látványos fejlődése tehát az adatmodellezés körül zajlott. Történt azonban más területen is egy s más. Három olyan momentumot szeretnék kiemelni, amely már a következő korszakot alapozta meg:

1. A strukturált programozás megjelenése.

2. A projektvezetés módszereinek kidolgozása.

3. A dokumentáció fontosságának felismerése.

A hardverfejlesztések eredményeképpen rendelkezésreálló egyre nagyobb központi és háttértárak egyre nagyobb programok megírását és futtatását tették lehetővé. Világossá vált, hogy a nagy programokat sokkal könnyebben és megbízhatóbban lehet elkészíteni, ha modulokra bontjuk őket és a modulokat összekapcsolva egy szerkezetet - *struktúrát* - alakítunk ki. Az is kiderült, hogy a struktúrálásra szabályokat lehet kidolgozni, így alakult ki a *strukturált programozás* elmélete és gyakorlata. E terület legjelentősebb szakemberei *Jackson, Warnier* és *Dijkstra.*

Az egyre fejlettebb adatkezelési és programozási alapokon, valamint az egyre nagyobb kapacitású gépeken egyre nagyobb információs rendszerek kidolgozása lett a feladat. Ezek már komoly beruházásoknak voltak tekinthetők és ennek megfelelő komolysággal kellett foglalkozni a velük kapcsolatos döntésekkel, a vezetésükkel. Kialakult tehát az általános projektvezetés egy sajátos területe, a *számítógépes rendszerfejlesztési projektek irányítása.* Ennek eredményeképpen egyre inkább finomodott a fejlesztés munkaszakaszainak, az azokon belüli tevékenységeknek a felbontása (életciklus), a közöttük levő logikai kapcsolatok *struktúrájának* meghatározása, a tevékenységek időigényének becslése.

A nagy rendszerek kialakítása hangsúlyozottabbá tette a dokumentáció fontosságát. Határozott törekvések történtek szabványos dokumentációs rendszerek kialakítására. Ezzel összefüggésben egyre sürgetőbben jelentkezett az igény a rendszerfejlesztés számítógépes támogatására. Nagyon nehéz volt ugyanis az egyre terjedelmesebb dokumentáció manuális kezelése. Technikailag már látszott, hogy az adatbázis kezelő rendszerek minden további nélkül alkalmazhatók egy sajátos adatbázis, az u.n. *fejlesztési adatbázis* kezelésére is. Akkoriban az ilyen tartalmú adatbázisokat *adatszótáraknak* neveztük. Az adatszótár tehát egyfajta számítógépesített dokumentációs rendszernek tekinthető.

Ennek továbbfejlesztését jelentik a napjainkban egyre terjedő *CASE* (Computer Aided Software Engineering = számítógéppel támogatott szoftver tervezés) *eszközök*, amelyek a rendszerfejlesztési munka számítógépes támogatását az egyszerű gépesített dokumentáción jóval túlmutató szinten kínálják.

A fent vázolt fejlődésnek szinte az első pillanatától természetes kísérője volt a rendszerfejlesztési munka irányítási módszereinek, az u.n. *projektvezetés*nek a fejlődése. Ebben a vonatkozásban is egyre inkább előtérbe került a számítógépes támogatás.

Mindebből úgy tűnhet, hogy a szakma nagyszerű fejlődését tudhatjuk magunk mögött, a kép azonban korántsem ennyire pozitív.

1.3 Gondok

A közelmúlt és a jelen szoftver- ill. információs rendszerfejlesztési projektjei a következő fontosabb gondokat, problémákat mutatták, sőt mutatják:

1. Hosszú, és a tervezettet legtöbbször meghaladó fejlesztési idő.
   Sajnos ma is ez a legáltalánosabb gond és sok felhasználó ezért már előre tart a rendszerfejlesztési projektek elindításától. Nehéz a munka megtervezése, mert nem normázható tevékenységeket kell elvégezni, de az a körülmény is hozzájárul a látszólag rossz tervezéshez, hogy a projekt megkezdése előtt gyakran nem világos az elérendő cél és a megoldandó feladatok.

2. Magas fejlesztési költségek, amelyekkel szemben kevés a kimutatható eredmény.
   A fejlesztési költségek jórészt az előző pontban említett, elhúzódó jelleg miatt válnak a vártnál nagyobbá. Ugyanakkor objektív nehézséget jelent a várható eredmények számszerűsítése. Gondoljunk pl. arra, milyen nehéz kimutatni, mennyi haszonnal jár egy vezetői döntés jobb megalapozottsága!

3. Az elkészült rendszerek általában nehezen módosíthatók, rugalmatlanok.
   A legnagyobb hibát ezzel kapcsolatban az jelenti, ha a gondot a *hiányos fejlesztési dokumentáció* okozza. Ennek az is lehet az oka, hogy az 1. pontban említett elhúzódást a legkönnyebben elhagyhatónak tűnő dokumentáció felületes, vagy hiányos elkészítésével próbálják ellensúlyozni. Később igen nehéznek - ha nem lehetetlennek - bizonyul a hiányosan dokumentált rendszer módosítása.

   A rugalmatlanságot *hibás tervezés* is okozhatja, akár a rendszer funkcióinak helytelen megválasztása, akár rossz programtervezés révén.

4. A felhasználói követelmények gyakran kielégítetlenül maradnak.
   Tulajdonképpen az itt felsorolt valamennyi hiba között ez a legsúlyosabb, mert azt jelenti, hogy igen gyakran rossz minőségű rendszerek készülnek. A jó minőség elsőrendű követelménye ugyanis az, hogy *a rendszer feleljen meg a vele szemben támasztott követelményeknek!*

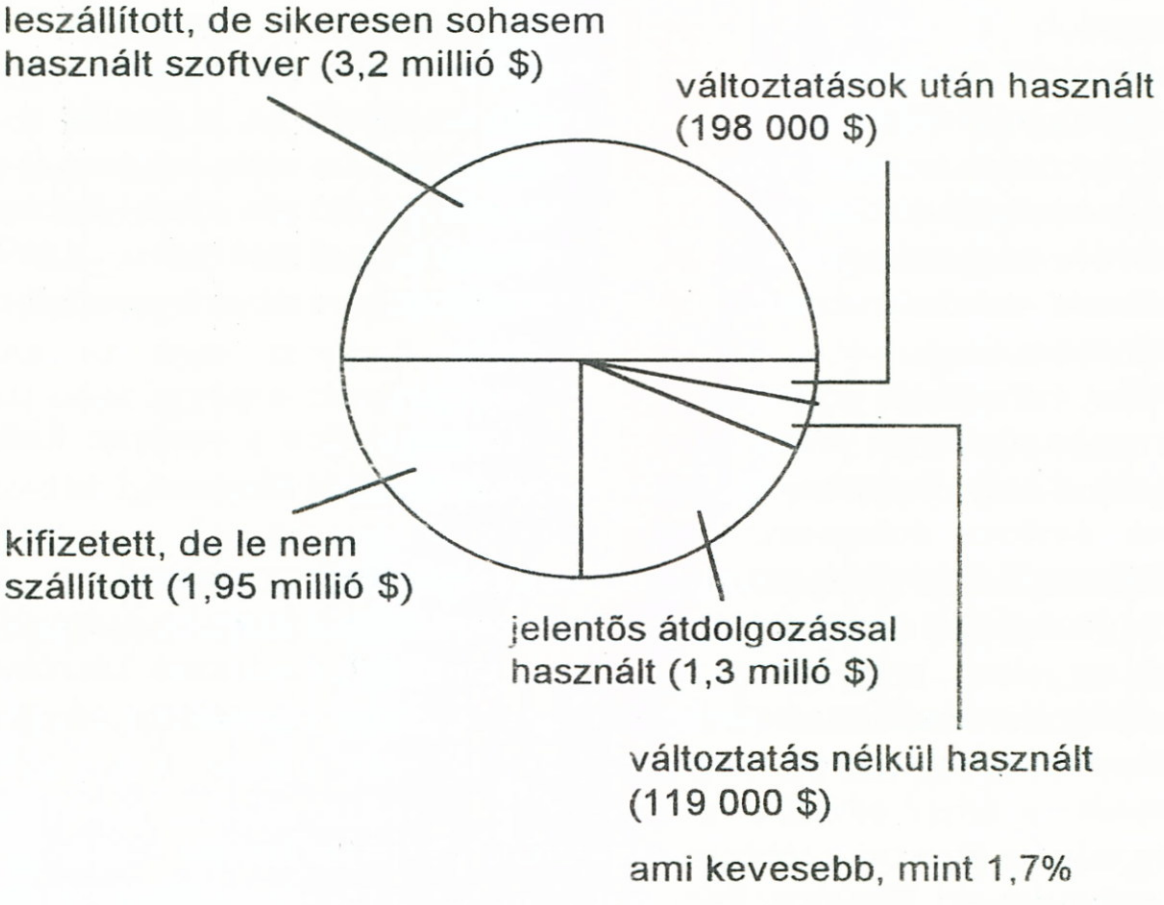
5. Még mindig gyakori a többszörös adattárolás (redundancia).
   A *redundancia* bizonyos foka nem káros, hanem elengedhetetlen (kifejtését ld. az adatmodellezés tárgyalásakor). Ha azonban a többszörös tárolásnak nincs a rendszer adatszerkezetét, vagy biztonságát meghatározó szerepe,

akkor felesleges, sőt káros. Egyértelműen tervezési hibát jelez, mint *tünet.* Valójában *azt jelzi, hogy baj van a rendszer integráltságával.*

6. Új eszközökön sokszor régi rendszer valósul meg.
Információs rendszerek fejlesztésekor többnyire az a cél, hogy egy létező rendszert - annak hibái és/vagy hiányosságai miatt lecseréljünk. Ez egyúttal igen sokszor módot ad azoknak az eszközöknek a korszerűbbekkel való helyettesítésére is, amelyeket a rendszer működtetésére kívánunk használni. Gyakori fejlesztési hiba, hogy az új rendszer kialakításakor nem veszik figyelembe az új eszközök sajátosságait, hanem a régi rendszerben szokásos megoldásokat mechanikusan átviszik az az új eszközökre. Ezáltal az eszközök kihasználása messze nem a kívánatos hatékonyságú lesz.

A felsorolt gondokra vezethető vissza, hogy viszonylag kevés volt a sikeres információs rendszerfejlesztési projekt.

Ezt a helyzetet jól jellemezte egy 1979-es amerikai felmérés eredménye (1. ábra)



1. ábra

A felmérés összességében 6,8 millió dollár értékű rendszerfejlesztési szerződésállományra terjedt ki.

Ha a fenti pontokba foglalt gondok közös okát akarjuk megfogalmazni, akkor azt mondhatjuk, hogy nem volt elég fejlett rendszerszervezési módszertanunk. Gondoljunk csak egy másik szakterületre, ahol szintén rendszereket kell létrehozni, a mérnöki tudományok területére! A munka ezen a területen sokkal pontosabban hajtható végre, a projektek világosabban átláthatók és irányíthatók. A mérnöki munkák esetében ugyanis már hagyománya van a jól meghatározott tervezési és kivitelezési *technológiák*nak. Ennek analógiáját kellett megteremteni a rendszerfejlesztés területén. A rendszerszervezési módszertan a mi szakterületünk sajátos technológiája, és a mérnöki munkával való hasonlósággal kapcsolatos utalás meg is jelent egyes korszerű módszertanok nevében is és a már említett CASE kifejezésben is. Ez pedig az angol "engineering" szó, amelyet csak a mérnöki tervezés megjelölésére használtak korábban.

2. fejezet
Strukturált módszertanok

A hetvenes évek végén nyolcvanas évek elején megszületett az a felismerés, hogy nincs itt különleges elsőbbsége sem a folyamatoknak, sem az outputoknak, sem az adatoknak - legalábbis ami a rendszerszervezési munka sorrendiségét illeti. Minden rendszerelemmel foglalkozni kell önmagában is, valamint egymással való kapcsolatukkal is. Mindegy tehát, hogy az elemzést, vagy a tervezést hol *kezdjük*. Ennél sokkal lényegesebb azt tudni, hogy a munka egy adott szintjén (pl. elemzés, vagy tervezés) minden szükséges tényezővel foglalkozzunk.

Az új módszertani törekvésekre jellemző, hogy a korábbi eredmények mind megmaradtak (persze folyamatosan tovább is fejlődtek). Ami a minőségi ujdonságot igazából hozta ebben a korszakban, az az elért eredmények rendszerbe foglalása. A strukturált rendszerszervezést tehát úgy is tekinthetjük, mint az addigi eredmények összefoglalását, egyfajta "*kassza-csinálást*".

Ennek során nagyon határozott törekvés érvényesült a rendszerfejlesztési folyamat *technologizálása* irányában. Egyes módszertanok ezt nevükben is igyekeztek kifejezni, mint pl. *James Martin* **Information Engineering**-je, amely a már említett mérnöki tervezéssel való analógiára utal.

A strukturált módszertanok néhány további jellegzetes képviselője a **SUMMIT-D** (*Coopers & Lybrand*), a **Method/1** (*Andersen Consulting*) és az **SSADM** (*CCTA, NCC*).

A 70-es, 80-as évek fordulója táján kifejlesztett rendszerszervezési módszertanokat, melyeknek a korábbiaknál jóval pontosabb a meghatározottságuk, dokumentációs rendszerük, egy szóval technológizáltságuk szintje, gyűjtő néven strukturált módszertanoknak nevezzük.

Vegyük sorra, - most kicsit részletesebben - mik a legfontosabb jellemzőik.

2.1 Termékszemlélet

Térjünk vissza egy kicsit a mérnöki munkával kapcsolatos hasonlóságra. A mérnök feladata valamilyen ipari termék, vagy létesítmény megtervezése. Ez

munkájának végeredménye/végterméke, amely természetesen nem egy lépcsőben áll elő, hanem részekből épül össze. Maguk a részek (pl. alkatrészek, szerelvények) is tekinthetők termékeknek. A végtermék tehát termékek hierarchiájaként is leírható, és ez az, amit alkatrész-, vagy darabjegyzéknek szoktak hívni. Tervezéskor először a kívánt végtermék tulajdonságait határozzák meg, majd az ehhez szükséges szerelvényeket, alkatrészeket. Az így meghatározott elemek részletes tervezése vezet el a kész tervhez.

Ezt a szemléletet érvényesítik a strukturált módszetanok az információs rendszerek, vagy szoftver csomagok tervezésekor.

Eszerint a fejlesztésnek nem csupán végterméke van (szoftver, vagy információrendszer), hanem minden tevékenység valamilyen terméket állít elő, vagy módosít (modelleket, diagramokat, specifikációkat, programokat stb.). Az előállítandó terméktípusok a módszertanban előre meghatározottak, és ami egy további nagyon lényeges jellemző: határozott *minőségi követelmények*nek kell eleget tenniük. A *minőségkezelés* tehát kifejezett hangsúlyt kapott.

2.2 A fejlesztés menetének pontos előírása

Ahogyan a műszaki technológiáknak megvannak a jól meghatározott fázisai, ugyanúgy a strukturált módszertanok is pontosan előírják a fejlesztés szakaszait, az azon belül szükséges tevékenységeket, azt, hogy milyen végeredményének - vagyis termékének - kell lennie az egyes tevékenységeknek, ehhez milyen más tevékenységek termékeit kell felhasználni, milyen technikákat kell használni az egyes termékek előállításához és milyen minőségi követelményeknek kell a termékeknek eleget tenni.

Ilyen módon maga a fejlesztés menete is mintegy szabványossá válik és a készülő termékek is szabványosított feltételeknek tesznek eleget. Ezáltal az egész munka sokkal áttekinthetőbbé válik, tehát pontosabban tervezhető és irányítható. Így csökkenteni lehet a pontatlan tervezésből származó későbbi határidő- és költségtúllépések valószínűségét.

Az előírt minőségi követelmények mércét szabnak a projektben résztvevők egyéni munkateljesítményének megítéléséhez. Ez tovább erősíti a projektvezetés megalapozását és hatékonyságát.

2.3 Technikák

A módszertan abban különbözik a módszertől, hogy míg az utóbbi egy jól körülhatárolt feladat megoldásának eszköze, addig a módszertan egymással összefüggő feladatok megoldásának egymással összefüggő, sőt egymásra épülő módszereit jelenti. A módszertan fogalma tehát tágabb, módszerek egységes rendszereként értelmezzük.

A strukturált módszertanokban használatos módszereket többnyire technikáknak szoktuk nevezni. Vannak olyan technikák, amelyeket ezekkel a módszertanokkal egyidejűleg fejlesztettek ki, de többségük már korábban is ismert volt valamilyen formában. Az újszerűséget éppen az említett rendszerezés jelentette, tehát az, hogy a különböző technikák egymásra épülnek.

Néhány példa a technikákra: adatmodellezés, folyamatmodellezés, dialógus-tervezés, esemény modellezés stb. Ezeknek a részletes tárgyalását a könyv további részei tartalmazzák.

2.4 Elemzés felülről lefelé, tervezés alulról felfelé

"A fától nem látja az erdőt" - szokták mondani egyes emberekre. Régen rossz, ha ezt rendszerszervezőre mondják. Arról van szó ugyanis, hogy amikor meg akarunk ismerni valamit - tehát elemezzük - akkor nem szabad az apró részleteknél *leragadnunk*, mert így kevés esélyünk lesz arra, hogy a lényeget *meg tudjuk ragadni*. A megismerendő rendszert tehát alrendszerekre, azokat funkciókra, majd folyamatokra stb. bontjuk le, amíg eljutunk a tovább már nem részletezendő szintig. Ekkor birtokában vagyunk azoknak az alkatelemeknek, amelyek módosítandók, vagy lecserélendők ahhoz, hogy az új rendszert megtervezzük.

Egy további fontos felismerés, hogy minden szinten belül egyaránt foglalkoznunk kell az információs rendszerek mindegyik alapvető elemcsoportjával: az *adatok*kal, a *folyamatok*kal (feldolgozások) és a rendszer *határfelület*eivel (input/output és feldolgozás/adatbázis interfész).

Az elemzéssel szemben tehát a tervezés alulról - a részletektől - felfelé haladva építkezik és így jut el az új egészig. Ennek oka az, hogy a módosított, vagy új alapelemek határozzák meg a felettük levő összeépítési szint elemeit és ez így követhető végig felfelé haladva.

2.5 Fizikai és logikai

Minden információs rendszer, vagy szoftvertermék esetében látnunk kell, hogy annak van egyfajta belső logikája - a felhasználó szempontjából ez a leglényegesebb - és van valamilyen konkrét megvalósítása. Ez utóbbi a rendszer belső lényegének adott eszközökön való tényleges megtestesülését jelenti.

A rendszerek fizikai és logikai vonatkozásainak szétválasztott kezelése jellemző a strukturált módszertanokban. Ennek oka, hogy ilyen módon csökkenteni lehessen az egyszerre kezelendő problémák számát, valamint az, hogy világosan elhatároltan meg lehessen fogalmazni a rendszerek logikai szintű leírását, amely hosszú ideig stabil maradhat, mert nem függ azoktól a konkrét eszközöktől, amelyeken adott esetben a rendszert működtetni kell.

2.6 Fokozatosság és iterativitás

A fokozatosság azt jelenti, hogy egy-egy tervezési objektummal kapcsolatban nem törekszünk arra, hogy amint elkezdünk vele foglalkozni, azonnal teljesen le is írjuk, meg is határozzuk. Pl. a fejlesztés kezdetén egy egyedtípusról többnyire csak a nevét, az azonosítóját, meg a kapcsolatait tudjuk. Később egyre több tulajdonságtípusát ismerjük meg, aztán pontosítjuk belső és külső szerkezetét, lehetséges állapotait, stb.[1] Ennek ellenére az egyedtípus dokumentálását megkezdjük a lehető legkorábbi állapotban, legfeljebb a kapcsolatos formanyomtatványnak csak néhány rovata lesz kitöltve. Aztán ahogy sokasodnak az ismereteink, úgy "hízik" a dokumentáció tartalma. Ez lehetővé teszi, hogy minden ismeret egységes módon a lehető legkorábban rendelkezésre álljon a munka valamennyi résztvevője számára.

Az iteráció fogalma a matematikából ismert és azt jelenti, hogy egy meghatározott műveletet elvégezve, majd a kapott eredményen ismételten végrehajtva egyre pontosabb eredményhez jutunk.

A rendszerszervezési munka esetében ennek az elvnek az alkalmazása azt jelenti, hogy az elvégzendő tevékenységek nem szépen sorban követik egymást. Nem mondhatjuk azt - sajnos - hogy amit egyszer elvégeztünk, az "le van tudva". A dolgok normális menetéhez tartozik, hogy amint haladunk valamely rendszer fejlesztésében, új ismeretekre teszünk szert, ami más megvilágításba helyezheti számunkra a korábban már megoldottnak hitt problémát. Vissza kell tehát térnünk és újra el kell végeznünk valamit, hogy így egy javított eredményhez

[1] Az itt szereplő fogalmak esetleg ismeretlenek. Kifejtésüket ld. a II. Részben.

jussunk. Hogy ez a helyzet egy adott projekt esetében mely tevékenységeknél fordul elő, azt nem tudhatjuk előre.

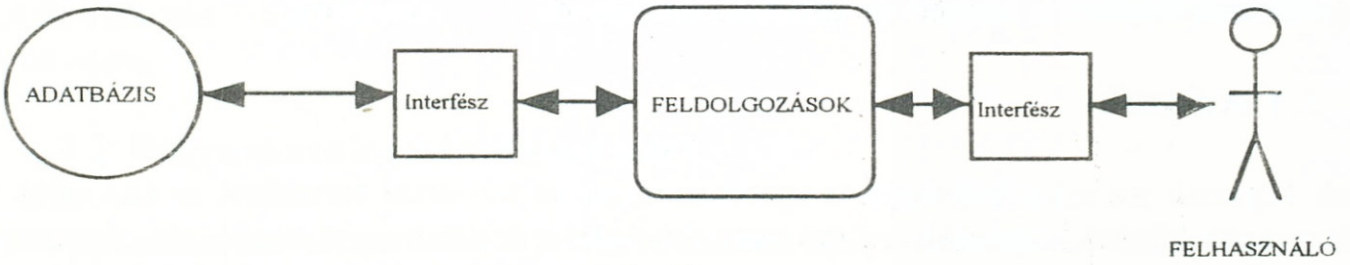
Ezek a fontos jellemzők a 70/80-as évtized-fordulóra érték el azt az érettséget, hogy módszertanokat lehetett alapozni rájuk.

A strukturált módszertanok megjelenését - amint arra már utaltunk - sok részterület fejlesztése előzte meg. Nem lehet tehát egyértelműen valakinek a nevéhez kötni ennek az új irányzatnak a megjelenését. Vannak viszont olyan szakemberek, akiknek a hozzájárulása igen jelentős volt, ilyenek pl. Bachman, Codd, Jackson, DeMarco, Yourdon, James Martin.

3. fejezet
Információs rendszerek összetevői

Kezdjük azzal, hogy vannak, akik különbséget tesznek *információrendszer* és *információs rendszer* között. Ennek a könyvnek nem célja, hogy akadémikus vitákban állást foglaljon, tehát benne a kétféle megnevezés szinonímaként fordul elő. Meggyőződésem, hogy nem ez a kérdés dönti el, tud-e valaki kiváló minőségű rendszert létrehozni, vagy nem.

Az információs rendszer különféle típusú elemekből épül fel és fontos, hogy az u.n. ember-gép rendszerek csoportjába tartozik. Hogy melyek a rendszert alkotó elemek csoportjai, az jórészt kiolvasható a 2. ábrából.



2. ábra

Az ábrán a ma korszerűnek tekinthető felfogást látjuk, melynek értelmében a rendszer egyik oldalán a felhasználó áll, a másik oldalán azok az adatok, amelyek a felhasználó számára viszonylag hosszú ideig fontosak. Ezek az adatok alkotják a szóban forgó információs rendszer u.n. adatbázisát. A felhasználó feldolgozásokon keresztül tart kapcsolatot ezzel az adatbázissal. A kapcsolattartás kétirányú, és a feldolgozás mindkét irányban lehet akár egyszerű, akár műveletvégzéssel egybekötött adattovábbítás is.

Ahhoz, hogy a felhasználó kapcsolatba kerülhessen a feldolgozásokkal, szükség van egy olyan rendszerelemre, amely mind a felhasználó, mind a feldolgozás felé a neki megfelelő alakban "tálalja" az adatokat. Ezt a rendszerelemet nevezzük *interfész*nek (az angol interface szó alapján). Magyarra a legjobban az "illesztés" szóval lehetne fordítani, de ez nem honosodott meg.

A feldolgozások és a tárolt adatok között is szükség van interfészre, mert legtöbbször az adatok egészen másként vannak tárolva az adatbázisban, mint ahogyan azokat a feldolgozások igénylik, ill. szolgáltatják.

Végül szót kell ejtenünk egy olyan elemcsoportról, amely a 2. ábrán nincs feltüntetve: az eszközökről. Azokról az eszközökről, amelyek az információs rendszerek megvalósításához a rendelkezésünkre állnak. Mint ismeretes, ezeket két csoportba sorolhatjuk: hardver és szoftver.

A rendszerek megtervezendő része általában az itt felsorolt elemcsoportok közül csak háromra terjed ki - legalábbis a szó szoros értelmében - mégpedig az adatokra, a folyamatokra és az interfészekre. A felhasználókat nem kell "megterveznünk", de - azon kívül, hogy véleményük a fejlesztés szinte valamennyi szakaszában döntő - bizonyos értelemben "formálnunk" kell őket. Ez a rendszer használatára, esetleg további, az információs rendszerrel összefüggő ismeretekre való kiképzést jelenti.

Hasonló - de nem teljesen azonos - a helyzet az eszközök vonatkozásában. Ezeket vagy adottságként kell figyelembe vennünk vagy módunkban áll megválasztani.

Vegyük most sorra a három megtervezendő rendszer összetevőt.

3.1 Adatok

Ezek képezik az adatfeldolgozás nyersanyagát és egyúttal termékét is (az adat és információ között a továbbiakban nem kívánunk különbséget tenni, de jegyezzük meg: információn általában az adatnál többet értünk, olyan adatot, amelynek jelentéstartalmát hasznosítani tudjuk). Az 1. fejezetben láttuk, hogy az adatok szerepének megítélése miképpen változott az idők folyamán. Jelenlegi felfogásunk szerint az egyes szervezetek (mint pl. vállalat) működését hosszabb távon leíró adatok köre viszonylag stabil, ezért érdemes modellben leírni ezeket az adatokat és a közöttük fennálló kapcsolatokat. Ilyen módon jutunk az u.n. logikai adatmodellekhez, amelyek a számítógépi adatbázisok, vagy hagyományos állományok megtervezésének a kiinduló pontjaként használhatók. A logikai adatmodellekben optimális átfedéssel fordulnak elő az adatok, tehát csak olyan mértékű az átfedés, amit a logikai szerkezet megvalósítása feltétlenül szükségessé tesz.

A felhasználó szemlélete - tehát az a kombináció, ahogyan pl. informálódása érdekében látni kívánja az adatokat - a tárolt szerkezettől eltér. Ez nagyon könnyen érthetővé válik, ha arra gondolunk, hogy egy közös adatbázisból milyen sokféle lekérdezés képzelhető el, amelyek eredményeképpen a felhasználók részben átfedő végeredményeket (outputokat) kapnak. Ugyanakkor azt már tudja a kedves olvasó, hogy az adatbázisban igen korlátozott átfedést engedünk meg.

Következésképpen szó sem lehet arról, hogy az adattárolás a mindenkori output-igényeknek 1:1 módon megfelelne.

Ez ugyanígy igaz a bemeneti oldalra is, tehát az adatok input szerkezetére.

Ezért van értelme az adatok három különböző csoportjáról beszélni, ill. ezek tervezésével foglalkozni. Ezek tehát:

- bemeneti adatok (inputok),
- kimeneti adatok (outputok) és
- tárolt adatok (adatbázis)

A bemeneti és kimeneti adatok a feldolgozások és a felhasználó közötti interfész *felhasználói oldalán* jelentkeznek (2. ábra). A *feldolgozás felőli oldalon*, valamint a feldolgozás és az adatbázis közötti interfésznek szintén a feldolgozás felőli oldalán a feldolgozás igényei szerint kell, hogy az adatok megjelenjenek. Ebben az alakban az adatok csak az adott feldolgozás futásának ideje alatt fordulnak elő, ezért szokás ezt az alakot *tranziens*, vagyis *átmeneti* szerkezetnek is nevezni.

3.2 Folyamatok (feldolgozások)

A folyamatokat a strukturált szemléletben úgy tekintjük, mint az adatok átalakítását végző rendszer összetevőket. Két nagy csoportjukat különböztetjük meg:

- a lekérdező (vagy visszakereső) és
- a karbantartó

folyamatokat. Az előbbiek eredménye a felhasználó számára szükséges információ. Mivel minden információs rendszer célja a felhasználó munkájának segítése, ezért megállapíthatjuk, hogy a visszakereső jellegű feldolgozás (amely tartalmazhat a visszakeresett adatokon végzett átalakítási műveleteket is) az információs rendszer legfontosabb része.

Az adatok karbantartására azért van szükség, hogy a visszakereséshez mindíg valósághű (naprakész) értékekkel álljanak rendelkezésre, tehát tükrözzék vissza a bekövetkezett változásokat. Az információs rendszer által kiszolgált szervezet életében bekövetkező változások, mint események kiváltják a szervezet adatainak megfelelő megváltozását. Ez az adat karbantartás lényege. Talán meglepő, de úgy is fogalmazhatunk, hogy a karbantartás a szükséges rossz kategóriájába tartozik.

Maga a "folyamat" kifejezés gyűjtőfogalmat takar. Attól függően ugyanis, hogy milyen részletezettségi szinten foglalkozunk az információs rendszer adatátalakítást végző részével, beszélni fogunk folyamatokról, elemi folyamatokról, funkciókról, vagy eljárásokról. A folyamatok szinonímájaként fordul elő a "feldolgozás" kifejezés. Ez a felbontás megfelelően követi a strukturált módszertanokkal kapcsolatosan az előző fejezetben megemlített felülről lefelé haladó elemzést és alulról felfelé haladó tervezést.

3.3 Interfészek

A felhasználói interfész felhasználói oldalának elsősorban a tartalmi vonatkozását fejezi ki az input és output. A formai vonatkozás maga a külső megjelenés (pl. képernyő kép), amit *felület*nek is szoktak nevezni. Ha ezen a felületen az adatok nem karakteresen, hanem képek alakjában jelennek meg, akkor beszélünk grafikus felhasználói felületről (GUI=Graphical User Interface).

Az interfész feldolgozási részt is tartalmaz, ami jelenthet ellenőrzést, hibakezelést és adatcsoportosítást, átrendezést.

Mindebből bizonyára érzékelhető, hogy az interfész önmagában is meglehetősen összetett objektum lehet.

4. fejezet
A rendszerszervezési munka menete

Az előző fejezetben áttekintettük az információs rendszerek összetevőit, hogy világos legyen, milyen típusú elemeket kell megtervezni és létrehozni, ill alkalmazni ahhoz, hogy kialakítsunk egy új információs rendszert.

Nyilvánvaló, hogy ez a munka időben zajlik és megfelelő irányítási tevékenység is kell, hogy kapcsolódjon hozzá.

Ahhoz, hogy valamilyen munkavégzést irányítani lehessen, legalább három tényezőt kell ismerni:

- mi az előállítandó eredmény,
- milyen részekre bontható fel a munka,
- milyen típusú munkaerőt igényel a feladat.

A munka részekre bontása azért fontos, mert - a tapasztalat szerint - minél kisebb egységekre tudjuk lebontani a feladatot, annál pontosabban határozható meg (ill. becsülhető) az elvégzéséhez szükséges idő és annál könnyebben értékelhető, ellenőrizhető az eredmény. Ugyanakkor azt is könnyebb meghatározni, hogy milyen szakértelemre van szükség az egyes részfeladatok elvégzéséhez.

Az információs rendszerek, ill. szoftvertermékek kialakításának menete manapság már eléggé pontosan meghatározott és elfogadott. Megértéséhez abból célszerű kiindulni, hogy ez a feladat egy nagyobb, általánosabb feladatkör részeként fogható fel, ez pedig a *probléma megoldás*. Akadhat olyan olvasó, akinek a számára meglepetés, ha a probléma megoldásról általában kívánunk beszélni hiszen minden probléma más és más, de legalábbis nagyon sok - ha nem végtelen számú - probléma csoport létezik.

Ez valóban így van, mégis felismerhető a megoldás egy igen általános menete:

A problémát először észre kell venni, fel kell ismerni ahhoz, hogy egyáltalán foglalkozni lehessen a megoldással.

Ezután tájékozódni kell a probléma részleteit illetően, valamint a megoldáshoz rendelkezésre álló erőforrások és egyéb környezeti feltételek tárgyában. Ekkor

már vázlatokat lehet kialakítani a számításba vehető konkrét megoldásokról és el lehet dönteni, hogy melyik látszik a leginkább kivitelezhetőnek.

A választott megoldást részletesen meg kell tervezni, majd következhet a kivitelezés, a terv megvalósításának munkája.

Végül az elkészített megoldás használatba vételével fejeződik be a munka.

Szemléletes példa lehet erre egy szerencsés hajótörés esete. A tengeren viharba került egy jacht és irányíthatatlanná vált. Lakatlan sziget környezetébe sodródott és egy hatalmas hullámmal kikerült a parti fövenyre, majd a vihar elcsendesedett és már nem jött akkora újabb hullám, amelynek a hátán visszakerülhetett volna a kis hajó a vízre. A négy utas és a hajó is sértetlen maradt, a probléma az volt, hogyan tegyék ismét vízre a járművet, amely mintegy 15 méterre került a közben kisimult víztükörtől. A hajó súlya messze meghaladta a négy utas fizikai erejét, tehát az szóba se jöhetett, hogy felemelik és visszaviszik a vízhez.

Maga a probléma ebben az esetben nagyon könnyen felismerhető volt, de vajon milyen megoldásokat lehetett elképzelni?

Lehetett volna pl. várakozni, hátha jön egy hajó, amelyről észre veszik őket.Ez nagyon sok időbe is telhetett volna. Abból kiindulva, hogy a szigeten egy kis erdő is volt, felvetődött a gondolat, hogy emelő rudakat kellene készíteni és azokat a hajótest alá feszítve, fokozatosan odébb tologatva lehetne visszajuttatni a hajót a vízre. Félő volt azonban, hogy a feszítés és a homokon való súrlódás tönkreteszi a hajótestet. Végül az látszott a legjobbnak, ha vastagabb fatörzsekből "görgőket" készítenek és ezeken gurítják óvatosan a hajót.

Megtervezték, hogy három fatörzset használnak majd, valamint két erős rúdra is szükség lesz a görgők elhelyezéséhez és a hajótest oldal irányú támogatásához.

Elkészítették a betervezett eszközöket, majd segítségükkel sikeresen megoldották a problémát.

A fentiek alapján a probléma megoldás általános sémája a következőképpen írható le:

1. A feladat felismerése és megfogalmazása.
2. A helyzet elemzése, megoldási változatok átgondolása.
3. A megoldás megtervezése.
4. A szükséges eszközök kivitelezése
5. Megoldás az elkészített eszközök használatával.

Miután ez a megoldási séma általános, arra a problémára is alkalmazható, amelyet egy információs rendszer, vagy egy szoftver kifejlesztése jelent. Célszerű azonban figyelembe venni ennek a tevékenységnek a sajátosságait, és ennek megfelelően alakítani a megoldás menetét.

Az egyik ilyen fontos tényező a költség. A rendszerfejlesztés költségei viszonylag magasak, következésképpen ez a feladat beruházási projektként fogható fel, amellyel kapcsolatban meg kell vizsgálni a várható költségeket, mielőtt egyáltalán el lehetne indítani a projektet. A feladat megvalósítása egyaránt függ a technikai kivitelezhetőségtől és a gazdaságosságtól.

A probléma felismerése rendszerint a rendszerfejlesztő szervezeten kívül történik és így az indulásnál a feladat adottnak tekinthető, még ha később pontosítandó is. Mindezek következtében az első tevékenységkör úgy módosul, hogy annak keretében a megvalósíthatóságot kell megvizsgálni.

Az elemzés, a tervezés és a kivitelezés tartalma az információs rendszerekkel kapcsolatos tényezőket foglalja magába, tehát az adatokra, folyamatokra és interfészekre irányul. Fontos, hogy - az ember/gép rendszer jellegének megfelelően - a felsorolt tevékenységkörök a rendszernek mind az emberi, mind a gépi oldalára kiterjedjenek.

További sajátosság az elkészült megoldás használatba vétele, ami itt egy várhatóan hosszabb időn át folytatott rendszeres használatot fog elindítani. Meg kell termteni tehát a rendszeres, fennakadásmentes működtetés minden feltételét, amit a rendszer bevezetésének nevezünk.

Mindezek alapján felsorolhatók az információs rendszerek (szoftver csomagok) fejlesztésének szakaszai:

1. Megvalósíthatósági vizsgálat
2. Rendszerelemzés
3. Rendszertervezés
4. Kivitelezés
5. Bevezetés a gyakorlati alkalmazásba

Egy megjegyzés azok számára, akik esetleg hiányolják a tesztelést, mint külön szakaszt: ennek tervezése a rendszertervezés, végrehajtása pedig a kivitelezés része.

Nagyon messzire vezetne, ezért túl részletesen nem lehet foglalkozni ebben a könyvben az információs rendszerek méretének és kivitelezhetőségének a kapcsolatával. Annyit azonban mindenképpen tudni kell, hogy a kivitelezhetőség ésszerű korlátai miatt gyakorlatilag minden szervezeten (vállalaton, intézményen) belül több, egymással összefüggő, de egymástól el is határolható információs rendszerrel kell és lehet számolni. Annak érdekében, hogy ezeknek a folyamatos korszerűsítése összehangoltan és a szervezet egésze érdekeit szem előtt tartva történjen, szükség van valamiféle keretre. Ez a keret egy hosszú távra szóló terv, amely kijelöli a fejlesztések sorrendjét és azokat a feltételeket, amelyeket be kell tartani. Manapság ezt a kerettervet *informatikai stratégia* névvel szoktuk illetni. Az egyes rendszerfejlesztési feladatok ebben a stratégiában vannak megfogalmazva, ill. abból levezethetők, de mindenképpen azzal összhangban kell, hogy legyenek.

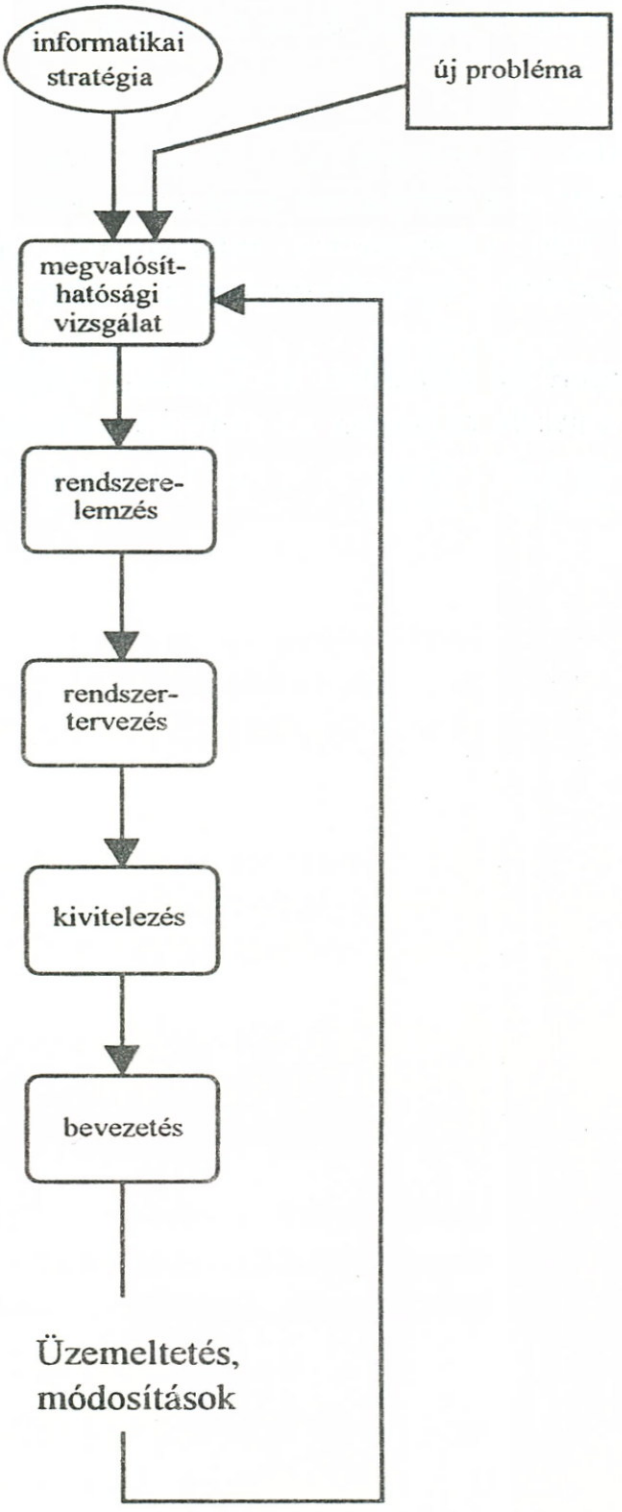
Az informatikai stratégia ilyenformán valamennyi konkrét fejlesztési projekt közös előzményeként is felfogható egy adott szervezeten belül.

Az egyes fejlesztési projektek azzal érnek véget - amint láttuk -, hogy megtörténik a bevezetés. Ettől a pillanattól megkezdődik a rendszer mindennapos üzemeltetésének időszaka. Rendszerfejlesztési szempontból két fontos tevékenység válik vagy válhat szükségessé ennek során:

Kb. féléves "éles" üzemelés után hasznos lehet kiértékelni az eltelt időszak tapasztalatait, hogy el lehessen végezni az esetleg szükséges korrekciókat, valamint hogy tanúságokat lehessen levonni a jövőbeli projektekre nézve. Ezt nevezik bevezetés utáni (poszt-implementációs) ellenőrzésnek.

A másik fontos tevékenység a rendszer folyamatos karbantartása, a valóságos élet követelményeihez igazodó módosítások elvégzése. A módosítás igénye törvényszerűen jelentkezik még a legjobb minőségben elkészített rendszerek esetében is, aminek az az oka, hogy az élet állandóan változik, és az információs rendszereknek követniük kell a változásokat.

A karbantartások természetesen nem érintik a rendszer alapelveit és amint telik az idő, gyorsan megjelennek újabb technikák, hardver és szoftver újdonságok.



3. ábra

Mindez oda vezet, hogy rendszerünk egyre "toldozottabb" megjelenésűvé és ugyanakkor erkölcsileg avultabbá válik. Végül eljutunk ahhoz a döntéshez, hogy teljesen új alapokra kell helyezni az adott információs rendszert, tehát új

fejlesztési projekt indul, előröl kezdődik a fejlesztési szakaszok sora. Úgy is fogalmazhatunk, hogy a kör bezárult, és ez az, amiért a *rendszerek életciklusáról* szoktunk beszélni.

A szokásos életciklusba a belépés a stratégiai terv megfelelő részének a sorrakerülése révén, vagy a stratégiában eredetileg nem szereplő, de valamilyen oknál fogva fontossá vált információs rendszerbeli probléma felvetődéséből történik. Az életciklus-modellt végeredményben a 3. ábra szerint írhatjuk le.

Az SSADM tárgyalásánál visszatérünk az életciklus fogalmához és megnézzük, mi a kapcsolata ennek a módszertannak az életciklus általános szakaszaihoz.

MÁSODIK RÉSZ

AZ SSADM SZERKEZETE

5. fejezet
Mi az SSADM?

Az SSADM rövidítés az angol Structured Systems Analysis and Design Method kifejezés kezdőbetűiből készült, és jelentése: strukturált rendszerelemzési és -tervezési módszer. Természetesen lehetne alkalmazni egy hasonló magyar rövidítést is (pl. SRETM) a módszer megnevezésére, azonban az SSADM annyira beivódott a nemzetközi szakirodalomba és gyakorlatba, hogy nem látszik sem célszerűnek, sem indokoltnak a megváltoztatása.

A "strukturált" jelző először a programozással kapcsolatban jelent meg a számítástechnikában (mostanában talán informatikának kellene neveznem) és elsősorban a Jackson, valamint Warnier által kidolgozott és publikált módszerek révén vált ismertté.

Az előző részben leírt, hátulról előre haladó fejlődésnek megfelelően, ezután a rendszerszervezés strukturáltnak nevezett módszere jelent meg, ami elsősorban Yourdon, Constantine és DeMarco nevéhez fűződik.

Ezzel - a többé-kevésbé eljárásorientált fejlődéssel - párhuzamosan alakult ki az adatmodellezés módszere az adatbázistervezéssel kapcsolatos kutatásokból Bachman, Codd, Date, Halassy és mások nevével fémjelezve.

A szakmában "örökzöld" probléma a dokumentáció, ami persze mindig szorosan összefügg magával az alkalmazott módszertannal - netán éppen annak hiányával. Ebben a tekintetben sokat segítettek az adatszótárak, újabban pedig a CASE eszközök.

A rendszerszervezési szakma további jól ismert gondja a megfelelő minőség biztosítása. Ilyen szempontból különösen érzékeny az elemzés és a tervezés szakasza, ahol nem lehetséges a tesztelés abban az értelemben, ahogyan ezt a programozásnál végezzük. Ugyanakkor az elemzési, ill. tervezési hibák hatása általában lényegesen nagyobb, kijavításuk pedig költségesebb, mint a programhibáké. A tesztelést valamiképpen helyettesítendő, itt is kialakult egy új módszer, az u.n. strukturált bejárás vagy csoportos átbeszélés.

A rendszerszervezés és a projektvezetés közötti határterületet képez a felhasználókkal való helyes kapcsolattartás, ezen belül is a fejlesztés alatt álló rendszerrel összefüggő felhasználói döntések előkészítése.

Erre elsősorban akkor van szükség, amikor több lehetséges megoldási változat közül kell kiválasztani azt, amelyik azután a további fejlesztési munka alapjává válik. Itt a fő kérdés az egyes változatok, valamint lehetséges következményeik világos megértetése a felhasználóval. Ezek a döntések nagyon fontosak a létrehozandó információs rendszer szempontjából, ezért szükséges volt egy sajátos döntéselőkészítési módszer kialakítása.

Az előző részben vázolt fejlődés eredményeképpen a 70-es évtized végére ezek a fent említett módszerek lényegében készen álltak, sőt tulajdonképpen használatban is voltak, bár finomításuk természetesen azóta is folyamatban van. Nem lehetett azonban világosan látni az összefüggéseiket, ill. az egymásraépülésük módját. Másképpen ezt úgy is fogalmazhatjuk, hogy *voltak strukturált módszerek a rendszerszervezés úgyszólván valamennyi szakaszára, nem volt viszont strukturált módszertan.* A hetvenes évek végére tehát már "a levegőben volt" egy strukturált rendszerszervezési módszertan, csak még meg kellett fogalmazni.

Ezt a munkát - másokkal nagyjából párhuzamosan - 1980-ban végezte el a brit *Learmonth and Burchett Management Systems (LBMS)* nevű cég, együttműködve a *Central Computer and Telecommunication Agency-vel (CCTA).* A megbízást a brit kormánytól kapták és a feladat az volt, hogy egy olyan módszertant alakítsanak ki, amelyet azután szabványnak lehet majd tekinteni a kormányintézményeknél folyó rendszerszervezési munkáknál. Az LBMS az új módszertant *LSDM* (LBMS Structured Development Method), vagyis az LBMS strukturált fejlesztési módszere néven hozta ki. A brit kormány a mószer tényleges szabvánnyá alakítását az NCC-vel (National Computing Centre) végeztette el, - ekkor kapta a módszertan az SSADM nevet - amely a CCTA-val karöltve ezt a szabványt azóta is rendszeresen karbantartja és fejleszti. 1990 óta van érvényben az SSADM 4. számú változata, amelyet magyar fordításban ajánlásként támogat az *Informatikai Tárcaközi Bizottság.*

Az SSADM a 80-as évek közepétől egyre népszerűbbé vált Nyugat-Európában és az Egyesült Államokban is, az alkalmazások köre pedig már nem korlátozódott a kormányintézményekre még Angliában sem. A népszerűségnek csak egyik oka a módszertan strukturáltsága és egyre következetesebb átgondoltsága. A másik ok feltétlenül az alacsony ár, ami bizonyára az állami szubvencióra vezethető vissza. Míg a konkurrens hasonló módszertanok ára a több tízezer angol font

nagyságrendben mozog, addig az SSADM-et leíró négy kézikönyv ára még póstaköltséggel együtt sem haladja meg a 200 fontot!

Persze senki sem gondolhatja komolyan, hogy ha megveszi a kézikönyveket, akkor már másnap nekifoghat a módszertan alkalmazásának. Elég komoly betanulási idővel kell számolni, ami magába foglalja egy megfelelő tanfolyamon való részvétel idején kívül a mindenképpen szükséges egyéni tanulást, valamint a kezdeti gyakorlati próbálkozásokra fordított időt. Sok függ az SSADM-et éppen mgismerni szándékozó szakemberek előképzettségétől és korábbi gyakorlati tapasztalataitól, továbbá az angol nyelvben való jártasságától. Ez utóbbi tényező a szakirodalom tanulmányozása szempontjából fontos. Mindent figyelembe véve a betanulási idő kb. 0.5-1.5 évet vehet igénybe a feltételek függvényében.

Minden módszertan tanulmányozásakor az egyik legfontosabb kérdés az életciklussal kapcsolatos: Hogyan épül fel, milyen szerkezetű a módszertan, és miért olyan, amilyen?

Az SSADM ebből a szempontból eléggé sajátos álláspontot képvisel, nézzük meg ezért, hogy milyen az életciklus-felfogása és hogy miképpen magyarázható ez a felfogás.

5.1 A minőség kihívása

A nagy számítógépes rendszerek kifejlesztése egyike a legösszetettebb vállalkozásoknak. Ilyen rendszerek létrehozása mintegy három évtizeddel ezelőtt kezdődött el, világszerte több tízezer (vagy százezer?) szakember vett részt a munkában, mégis azt kellett látnunk, hogy - legalábbis ehhez képest - viszonylag kevés sikerrel (ld. 1. ábra).

Ez a szomorú mérleg is nagyban ösztönözte a jobb minőségű információs rendszerek létrehozását biztosító módszertanok kifejlesztését.

Dehát mit is jelent valójában a *minőség* fogalma információs rendszerekre vonatkoztatva?

A szakemberek általában egyetértenek abban, hogy a jó minőségü rendszer

- pontosan azt szolgáltatja, amit előzetesen elvárnak tőle, ill.
- megfelel az eredeti specifikációnak.

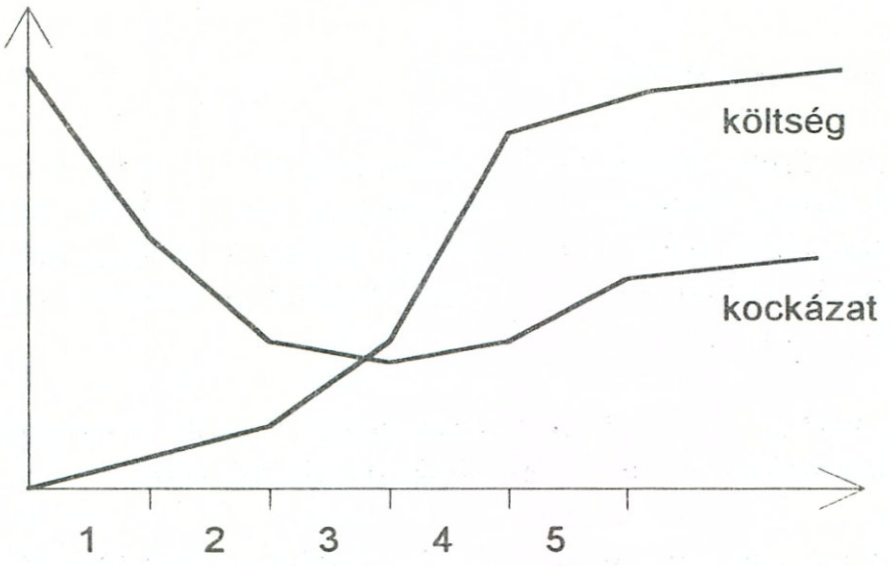
A minőség javítási lehetőségeinek kutatása során érdemesnek látszott összefüggést keresni a rendszerszervezési munka szakaszaival. A kérdés az, hogy vajon

valamennyi munkaszakasz azonos módon járul-e hozzá a követelmények pontos meghatározásához. Úgy is feltehető a kérdés, hogy a különböző szakaszokban elkövetett hibák azonos súlyúak-e.

Eléggé közismert, hogy a rendszerszervezési munka u.n. életciklus szerint történik, melynek részei a szakaszok/modulok. Ezek elhatárolása az egyes módszertanokban némileg különbözik, de ennek e könyv szempontjából nincs különösebb jelentősége. Vegyünk egy teljesen általános életciklus-felosztást, amely nem kötődik meghatározott módszertanhoz:

- 1. Megvalósíthatósági vizsgálat
- 2. Rendszerelemzés
- 3. Rendszertervezés
- 4. Kivitelezés
- 5. Bevezetés

Ha most megnézzük, hogyan alakulnak a rendszerfejlesztés költségei, ill.a bukás (a rossz minőségü rendszer létrehozásának) kockázata az idő függvényében, akkor a 4. ábra szerinti összefüggés állapítható meg.



4. ábra

Az ábrán az időt a fenti rendszerszervezési szakaszok egymásutániságával "mérjük". Ez persze nem eredményez egyenletes skálát, viszont érdekes következtetések levonására ad alkalmat:

- a költségek viszonylag mérsékeltek a korai szakaszokban;

- a rossz minőségű rendszer kifejlesztésének kockázata éppen ezekben a szakaszokban a legmagasabb, minimumát a kivitelezés során éri el, majd ismét emelkedik a bevezetésnél.

Ebből az az egyszerű következtetés vonható le, hogy a minőség javítását a leghatékonyabban az első három fejlesztési szakaszban folytatott munka módszereinek javításával, ill. az ott alkalmazott minőségellenőrzéssel érhetjük el.

Jogosan vetheti fel valaki a kérdést, hogy miért éppen úgy alakul a kockázati görbe, ahogyan az fel van tüntetve. A nagyon egyszerű ok az emberek - elsősorban felhasználók és rendszerszervezők - közötti kommunikációs igény mértéke. A kommunikációra a legnagyobb szükség az elemzés és a tervezés során van, éppen a feladat részleteinek a pontos megfogalmazása, továbbá a fejlesztők javaslatainak a felhasználóval való megértetése céljából. Ezt a folyamatot jelentősen befolyásolja az a körülmény, hogy a legtöbb esetben egymástól távol álló fogalmakban gondolkodik a két fél, olykor szinte más nyelven beszél! Ebből vezethető le a rossz feladat megfogalmazás, és ha ez valóban bekövetkezik, akkor már hiábavaló a legbrilliánsabb programozás a kivitelezés során, az elkészülő rendszer mindenképpen rossz lesz. A bevezetés során ismét fokozódik a felhasználók és rendszerszervezők között a kommunikáció és ez újabb félreértések forrása lehet.

Fontos azt is látnunk, hogy a programozás területén érvényesült szinte már a kezdet kezdetétől a legalaposabb minőségellenőrzés, a tesztelés, amit maga a számítógép tett lehetővé. Ugyanakkor a legkevésbé ellenőrzött és leglazábban kezelt szakasz mindig is az elemzés és tervezés volt. Mindebből következett az SSADM megalkotói számára az a tanulság, hogy az új módszertannak ezekre a szakaszokra kell helyeznie a fő súlyt.

5.2 Az SSADM helye az életciklusban

Szándékosan nem úgy fogalmaztam, hogy "az SSADM életciklusa", mert ez a módszertan nem fedi le a teljes fejlesztési ciklust. Sokat lehetne vitatkozni azon, hogy ez mennyiben jó vagy rossz, de most nem ez a cél. Fogadjuk el adottságként, hogy az SSADM az életciklus első három szakaszára terjed ki, tehát nem foglalkozik a kivitelezéssel és a bevezetéssel. Ennek alapvetően két oka van:

- az elemzést és a tervezést tartja a végtermék minősége szempontjából döntőnek;
- a kivitelezéstől kezdve olyan mértékben meghatározó az adott hardver/szoftver környezet, hogy arra igen nehéz általános érvényű módszereket adni.

Természetesen az az igazi, ha a teljes életciklusra rendelkezünk módszertani szabvánnyal. Mi tehát a megoldás?

A legkézenfekvőbb, hogy házilag "hosszabbítjuk meg" az SSADM-et a saját hardver/szoftver környezetünk ismeretében.

Egy másik igen érdekes megoldás, ha az SSADM-et kombináltan alkalmazzuk olyan módszertannal, amely szerkezetét illetően lefedi a teljes ciklust, de az u.n. technikákkal kevéssé foglalkozik. Ilyen pl. Coopers & Lybrand SUMMIT-D-je, amely fel is kínálja ezt a kombinációt.

Végül úgy is lehet dönteni, hogy nem kell az SSADM és vagy saját módszertant alakítunk ki, vagy másikat vásárolunk. Az első nagyon kockázatos, a második nagyon drága változat.

5.3 Három dimenzió

Az igen pontos ábrázolás igénye először a műszaki rajzban jelent meg. Itt fejlesztették ki azt a technikát, amely szerint a tér három dimenziójának megfelelően három különböző nézetben ábrázolva egy legyártandó tárgyat, a térbeli kép pontosan helyreállítható.

Valami hasonló megoldást kellett találni az olyan absztrakt "tárgyak" "legyártásához", mint amilyenek az információs rendszerek.

Az SSADM-ben ehhez a következő három "dimenziót" határozták meg:

- adatok,
- folyamatok,
- idő.

Mindegyik dimenzió kezelése u.n. *technikák* révén valósul meg. Ezek azok a módszerek, amelyek összehangolt rendszere jelenti a módszertant. Az összehangolást a módszertan u.n. *szerkezete* valósítja meg, amely megadja, hogy melyik tevékenységet mikor kell elvégezni. Az alábbiakban néhány ide kapcsolódó technikát sorolunk fel.

Az adatokhoz kapcsolódó technikák: *logikai adatmodellezés, relációs adatelemzés és fizikai adattervezés.*

A folyamatokhoz kapcsolódó technikák: *adatfolyam modellezés, funkciómeghatározás, I/O tervezés, dialógustervezés, menütervezés, logikai adatbázis folyamatok tervezése, fizikai folyamattervezés.*

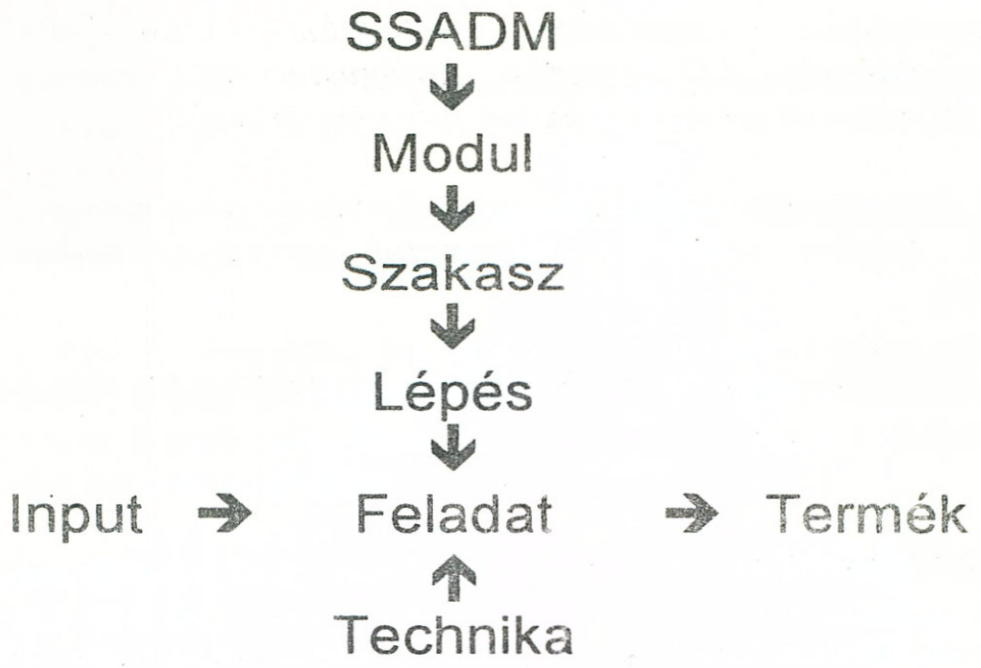
Az idővel kapcsolatos technikák: *egyedtörténeti diagramok, esemény-hatás diagramok, logikai adatbázis folyamatok tervezése, fizikai folyamat- és adattervezés.*

Ezekről a technikákról és összefüggéseikről bővebben a további fejezetekben lesz szó.

5.4 Az SSADM szerkezete

A strukturált módszertanok kialakulásával kapcsolatosan az előző fejezetben, valamint az SSADM-mel kapcsolatban a fentiekben elmondottak vezettek a módszertan szerkezetéhez. Maga ez a szerkezet is hierarchikus felépítésű, ilyen értelemben tehát önmagára nézve is követi a felülről lefelé haladó (top-down) szemléletet. Az SSADM egésze *modulokra*, a modulok *szakaszokra*, ezek *lépésekre*, a lépések pedig *feladatokra* vannak felosztva.

Azok a *technikák*, amelyekről az előzőekben szó volt, a feladatokhoz kapcsolódnak, közölve, hogy pontosan milyen módszerrel kell a feladatot végrehajtani, és ennek során milyen inputból milyen output *termék* állítandó elő (ld. *termékszemlélet*, kifejtve az előző részben). Az SSADM ilyen értelmű hierarchikus felépítését mutatja be az 5. ábra.



5. ábra

Az első szinten helyet foglaló modulok, ill. az ezek alá rendelt szakaszok a következők:

- **Megvalósíthatóság-elemzés**
  0. Megvalósíthatóság eldöntése

- **Követelmény-elemzés**
  1. Jelenlegi helyzet vizsgálata
  2. Rendszerszervezési változat kiválasztása

- **Követelmény specifikáció**
  3. Követelmények meghatározása

- **Logikai rendszerspecifikáció**
  4. Rendszertechnikai változat kiválasztása
  5. Logikai rendszertervezés

- **Fizikai rendszertervezés**
  6. Fizikai rendszertervezés

Az SSADM tehát öt modulból és hét szakaszból áll. A modulok nincsenek számozva, a szakaszok számozása pedig nullával kezdődik. Ennek oka, hogy a

megvalósíthatóság elemzésével foglalkozó szakasz nem kötelező az SSADM-ben. Ha tehát elhagyják, a szakaszok számozása még mindig 1-gyel kezdődik.

További érdekességet jelent a 3. szakasz. Előtte elemzés van, mögötte tervezés. Akkor ez mi? Hát ez az átmenet! Könnyű belátni, hogy az elemzés és a tervezés nem választható el mereven. Vannak tevékenységek, amelyekről nehéz eldönteni, hogy hová sorolandók. Nagyon jó példa erre az adatmodellezés, amely elemzés is, meg tervezés is egyszerre. A hetvenes években már ismertük ennek a szakasznak az analógiáját, úgy hívtuk: előtervezés.

5.5 Projektvezetés és minőségbiztosítás

Sokan azzal vádolják az SSADM-et, hogy túlságosan terjedelmes (már ami a dokumentációt illeti) és ezért nehézkes a használata. Kétségtelen, hogy a módszertan nagyon alapos dokumentációt ír elő, de meglehet, hogy akik vádaskodnak, megfeledkeznek egy nagyon lényeges pontról. Tévedés azt hinni, hogy egy SSADM-projekt lebonyolításának megkezdésekor elő kell vennünk a kézikönyveket, és szépen végre kell hajtanunk valamennyi ott szereplő feladatot A-tól Z-ig.

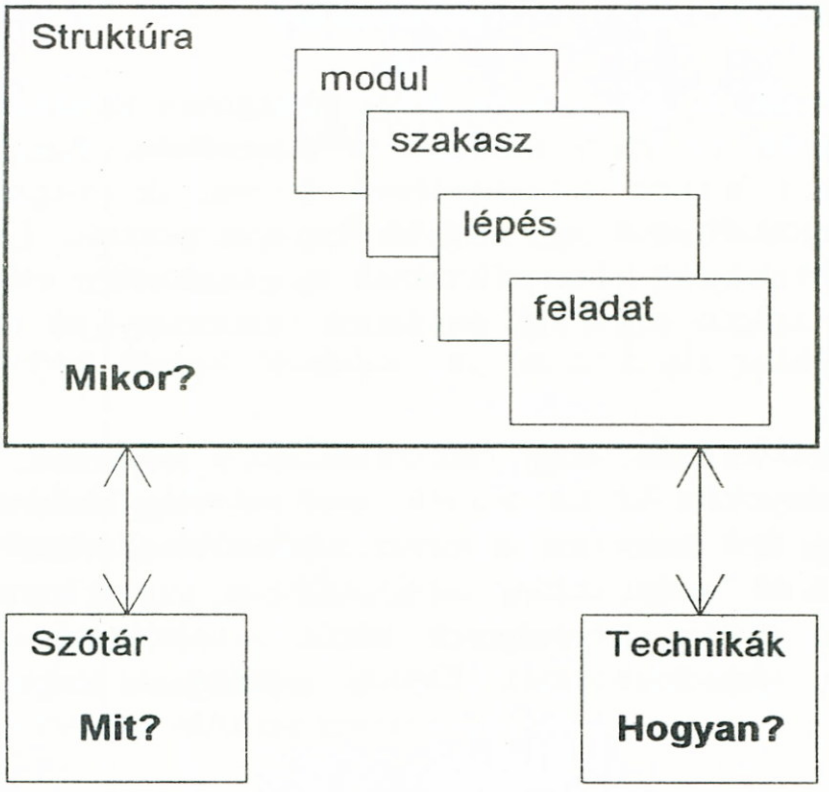
E helyett azzal kell kezdeni, hogy elkészítjük azt a tervezetet, amely szerint a projekt végeredményeként kívánt termék (szoftver, vagy információs rendszer) felépül, tehát meg kell határozni a menet közben előállítandó résztermékeket. Ekkor módunkban áll "testre szabni" az SSADM-et, vagyis kiválogatni azokat a résztermékeket az elvileg lehetségesek közül, amelyek *az adott projekthez szükségesek* (u.n. terméklebontás). Ezáltal csökkenteni lehet az elvégzendő munkát.

Ebben a projekt tervezési munkában nyújt alapvető segítséget az SSADM u.n. *adatszótára*, amely pontosan specifikálja a fejlesztési munka során előállítandó összes lehetséges terméket, a hozzájuk tartozó minőségi követelményekkel együtt.

A projekt időbeli lefutásának megtervezéséhez és az operatív irányításhoz az SSADM-projektekhez tartzó vezetési módszertant, a PRINCE-et célszerű alkalmazni. Ezt külön kézikönyvek írják le.

5.6 Összefoglalás

Az SSADM olyan strukturált rendszerszervezési módszertan, amely a fejlesztés elemzési és tervezési fázisait támogatja, és eleget tesz a strukturált módszertanokkal szemben támasztható valamennyi követelménynek. Felépítésében három nagyobb részt tartalmaz, ahogyan azt a 6. ábrán látjuk. *Strukturális* része az elvégzendő tevékenységek időbeliségével foglalkozik, *technikai* része azt mondja meg, hogyan kell a tevékenységeket elvégezni, *adatszótára* pedig leírja az előállítandó termékeket.



6. ábra

6. fejezet
Tervezés SSADM-ben

*Az SSADM megértéséhez a magyarázatot a kitűzött végeredménynél kezdjük és bemutatjuk, hogy ez a végcél (vagyis a **fizikai terv**) hogyan jelölte ki a hozzá vezető utakat. Levezetjük a fizikai tervezéshez szükséges inputokat, melyek a logikai specifikáció outputjai. Innen jutunk vissza a logikai tervezés által megkívánt inputokhoz. Eközben kitérünk a fontosabb tervezési lépésekre is.*

Az előző részben az olvasó képet kapott az SSADM fő szerkezeti felosztásáról, moduljairól és szakaszairól. Ennek alapján - egy kis egyszerűsítéssel - azt mondhatjuk, hogy a modulok közül kettő elemzéssel (Követelmény elemzés, Követelmény meghatározás), kettő pedig tervezéssel (Logikai rendszerspecifikáció, Fizikai rendszertervezés) foglalkozik. Ahhoz, hogy ezeknek a moduloknak, a bennük foglalt szakaszoknak és lépéseknek a szerepét, értelmét megérthessük, a legcélszerűbb a kívánt végeredményből kiindulni. Ez a végeredmény a tervezési tevékenység eredménye, ezért foglalkozunk ebben a részben a tervezéssel.

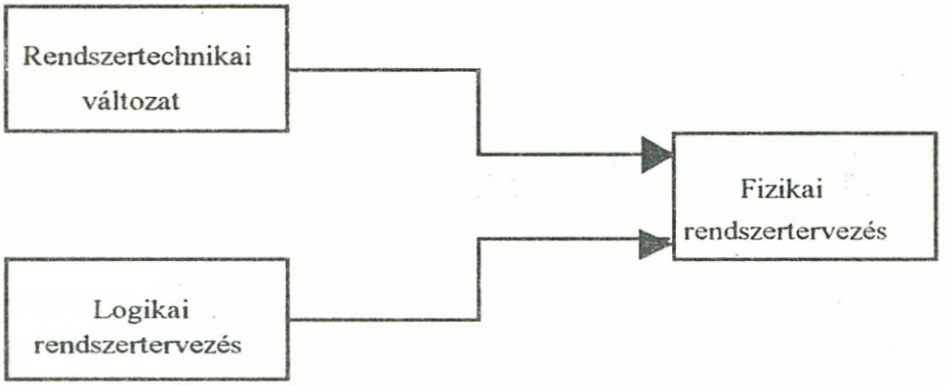
A fent említett két tervezési modul három fejlesztési szakaszt jelent, mert a *Logikai rendszerspecifikáció* két szakaszból áll:

- *Rendszertechnikai változat kiválasztása* és
- *Logikai rendszertervezés.*

Ehhez jön az utolsó modul egyetlen szakasza:

- *Fizikai rendszertervezés.*

Ezek a szakaszok nem egyszerűen sorakoznak egymás után, hanem határozott logikai szerkezetet alkotnak, amint az a 7. ábrán látható.



7. ábra

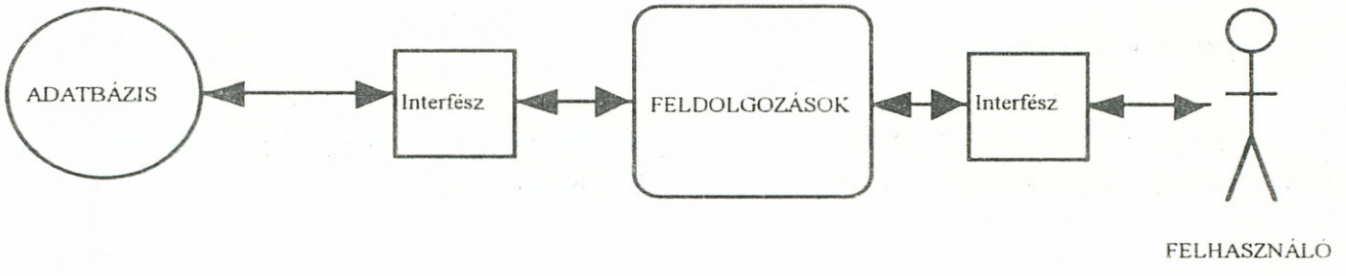
Ahhoz tehát, hogy a fizikai rendszertervezés elkezdődhessen, már készen kell lennie a logikai tervnek, továbbá meg kellett, hogy történjen a technikai környezet kiválasztása. E feltételek nélkül ugyanis nem tudnánk, *minek* a fizikai tervére van szükség és *mire*, milyen környezetre készítsük el a tervet.

6.1 A fizikai tervezés

A fizikai terv *célja*, hogy programtervezésre, ill. programozásra (program generálásra) alkalmas módon írja le a megvalósítandó rendszert (szoftvert). Három nagyobb csoportba sorolhatók azok a termékek, amelyeket ennek során létre kell hozni:

- adattervek,
- feldolgozástervek és
- interfész-tervek.

Azért ezek, mert SSADM-felfogásban az információs rendszerek alapszerkezete a következő (8. ábra):



8. ábra

A felhasználó és az adatbázis között helyezkednek el a feldolgozások, amelyek mind az adatbázishoz, mind a felhasználóhoz interfésszel csatlakoznak.

Kezdjük azzal, hogy mivel *nem* foglalkozik az SSADM: a felhasználói interfészek fizikai tervezésének módszereivel. Előírja a képernyők, input nyomtatványok, nyomtatási képek fizikai szintű (tehát konkrét) megtervezésének szükségességét, de nem ad ezekre semmilyen sajátos módszert mert e módszerek régóta ismertek és kiforrottak a szakmában. További indok, hogy az SSADM támogatja a *prototipizálás* alkalmazását, melynek során éppen ennek az interfésznek a megtervezése kap különös hangsúlyt, ugyanakkor az ilyen típusú tervezés részletes módszere jórészt a prototipizáló eszköztől függ.

Annyit azonban megállapíthatunk a felhasználói interfészekkel kapcsolatban, hogy a fizikai tervezés megkezdésekor ismerni kell a feldolgozások *I/O szerkezetét*, a *menüszerkezet*et és a megvalósítandó *dialógusok*at, beleértve ezeknek a tervezési objektumoknak a konkrét adattartalmát. Mindezek az elemek a logikai tervezés által szolgáltatandók. Emellett persze azt is tudni kell, hogy milyen *technikai eszközök*re számíthatunk a felhasználói interfész fizikai megvalósításához, ami a választott rendszertechnikai változat révén válik adottá.

Vegyük ezekután sorra a többi tervezési elemet, amelyek elkészítéséhez az SSADM sajátos módszereket kínál.

A fizikai adattervezés két lépésben történik:

- első közelítésű terv
- finomított terv

Az első közelítésű terv kialakításánál csak olyan, u.n. *ökölszabályok*at alkalmazunk, amelyek minden fizikai megvalósításnál érvényesek. A cél az, hogy a logikai tervezés egyik végeredményét jelentő logikai adatmodellt fizikai adattervvé alakítsuk át. Nagyon lényeges azonban azt tudni, hogy maga a *logikai adatmodell* - ill.általában a rendszer logikai terve - ettől még *megmarad* és alkalmas arra, hogy a jövőben egy másik fizikai környezetre való áttéréskor csak a fizikai tervezést kelljen újra elvégezni! Ezt a sajátosságot különösen a figyelmébe ajánlom azoknak a kollégáknak, akik egy pillanatnyi előny - a látszólag gyors előrehaladás - érdekében, vagy felhasználói nyomásra hajlamosak néhány soros, vagy mondatos felhasználói igény alapján azonnal a PC-hez ülni és programokat írni. Így sem integrált, sem jól karbantartható rendszert nem lehet készíteni!

Az első közelítésű terv elkészítésénél azt feltételezzük, hogy a logikai adatmodell egyedtípusaiból *rekordtípusok* (relációs táblák) lesznek, a rekordok tárolása *fizikai blokkok*ban történik, ahová az elérési igényeknek megfelelően csoportosítjuk és soroljuk őket. Ehhez szükség van annak ismeretére, hogy az információs rendszer egyes funkciói mely rekordokat igénylik elsődleges kulcs, melyeket másodlagos kulcs alapján. Ezeket a kiinduló információkat tehát szintén szolgáltatnia kell a logikai tervnek.

Az első közelítésű fizikai terv elkészítése módot ad arra, hogy a várható tárolóhely foglalásra vonatkozóan becsléseket végezzünk.

Viszonylag kevés esély van arra, hogy ez a terv már végleges is lehet, ezért van szükség a finomításra, amit optimálásnak is szoktak mondani.

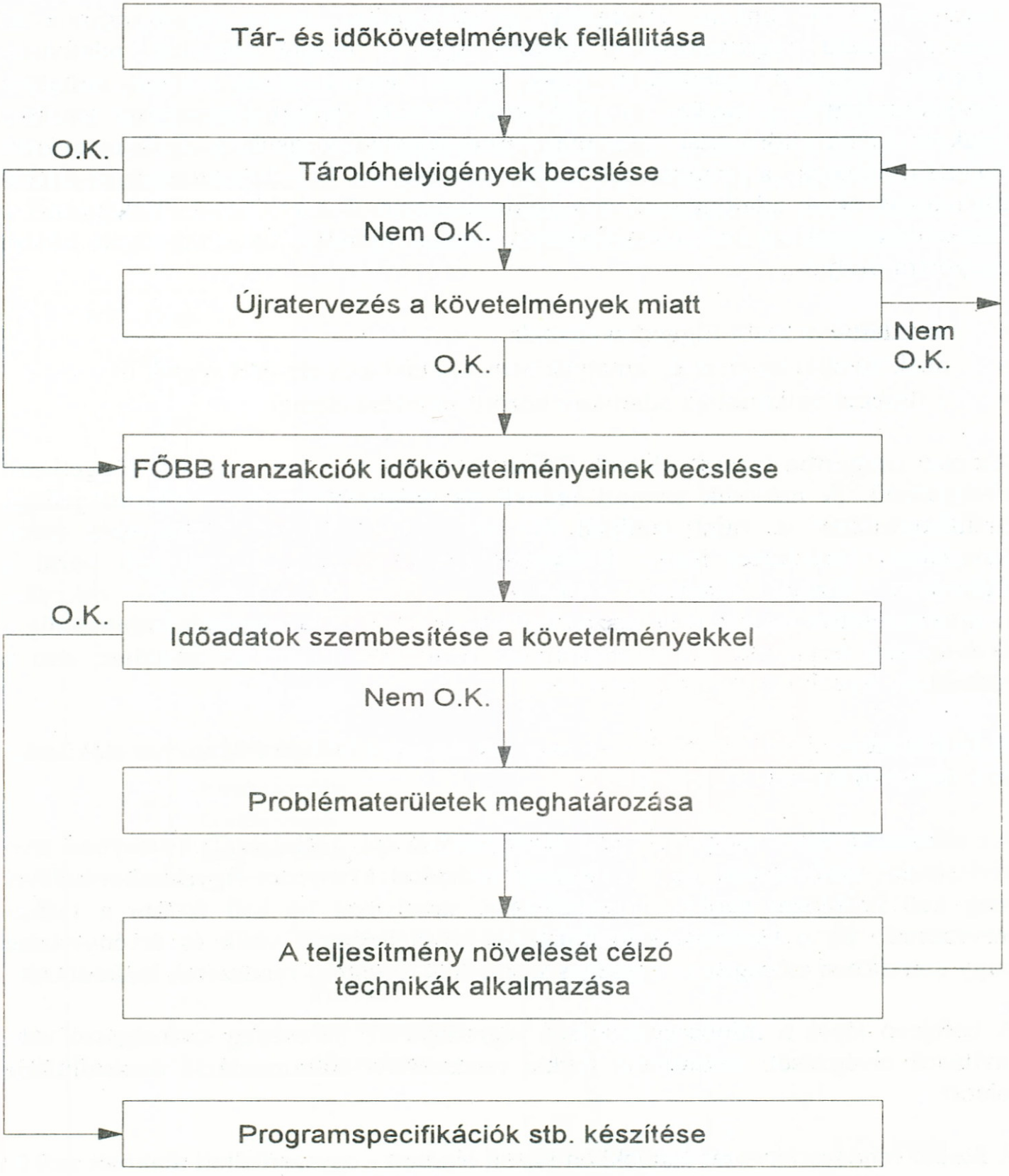
A finomításhoz a tárolóhely foglalásra, valamint a visszakeresésekre vonatkozó idő-követelmények ismerete szükséges. Ezeket szintén a logikai tervnek kell szolgáltatnia, vagy legalábbis közvetítenie.

Maga a finomítás a 9. ábra szerinti iterációval készül el.

A *feldolgozások fizikai tervezése* is két lépésre oszlik:

- funkcióelemek feltérképezése[2]
- funkciómeghatározás teljessé tétele

[2] Function Component Implementation Map (FCIM)



9. ábra

Már az eddigiekből is kiolvasható, hogy az SSADM az információs rendszerek eljárási, feldolgozási részét *funkciókra* osztja fel, amire a későbbiekben még

visszatérünk. A funkció lényegében gyűjtőfogalom, amely összekapcsolja az azonos feladat végrehajtásához tartozó input és output adatokat, adatbázis-részeket, valamint eljárásokat. Ezek között vannak olyan elemek, amelyek meghatározhatók a logikai tervezés során, mások viszont szükségessé teszik a konkrét fizikai környezet ismeretét, ezért meghatározásuk a fizikai tervezés feladata. Ilyenek: a szintaktikus hibák kezelése, az I/O formátumok, a fizikai dialógusok és az adatbázissal való kommunikáció. Fel kell tehát "térképezni" a funkciókat ezeknek az elemeknek a szempontjából. Az eredményt három csoportra osztjuk:

- az adattervezéstől független elemek,
- azok az eljárási elemek, amelyek nem az adatbázis elérését végzik és
- a funkció valamint az adatbázis közötti interfész elemei.

Az első csoportba tartozó elemek fizikai tervezése az adattervezéstől függetlenül elvégezhető. A második csoport igényli az adattervezésnek az eljárási jellegű melléktermékeit is, mint amilyen pl. egy, - a finomított adattervezés során bevezetett - rendezési igény. E csoport tervezésével tehát meg kell várni a finomított fizikai adatterv elkészültét. A *feldolgozás/adat interfész* megtervezéséhez viszont már készen kell lennie mind az adattervnek, mind pedig a feldolgozási terveknek. Ez a tevékenység nem tartozik sem az egyikhez, sem a másikhoz, hanem egy külön lépést képez.

A fizikai tervezéssel kapcsolatosan még két dologról kell szót ejteni: az előkőszítő és a befejező lépésekről.

Az előkészítő lépés során az adott környezetben (pl. vállalatnál) érvényben levő kivitelezési szabványok, és a választott technikai környezet figyelembevételével meg kell határozni azokat a szabályokat, amelyeket be kell tartani a fizikai tervezésnél. Ez a továbbiakban minőségi követelménnyé válik és érvényesítése nagy mértékben elősegíti az egységes szemléletű, integrált rendszerek kialakítását.

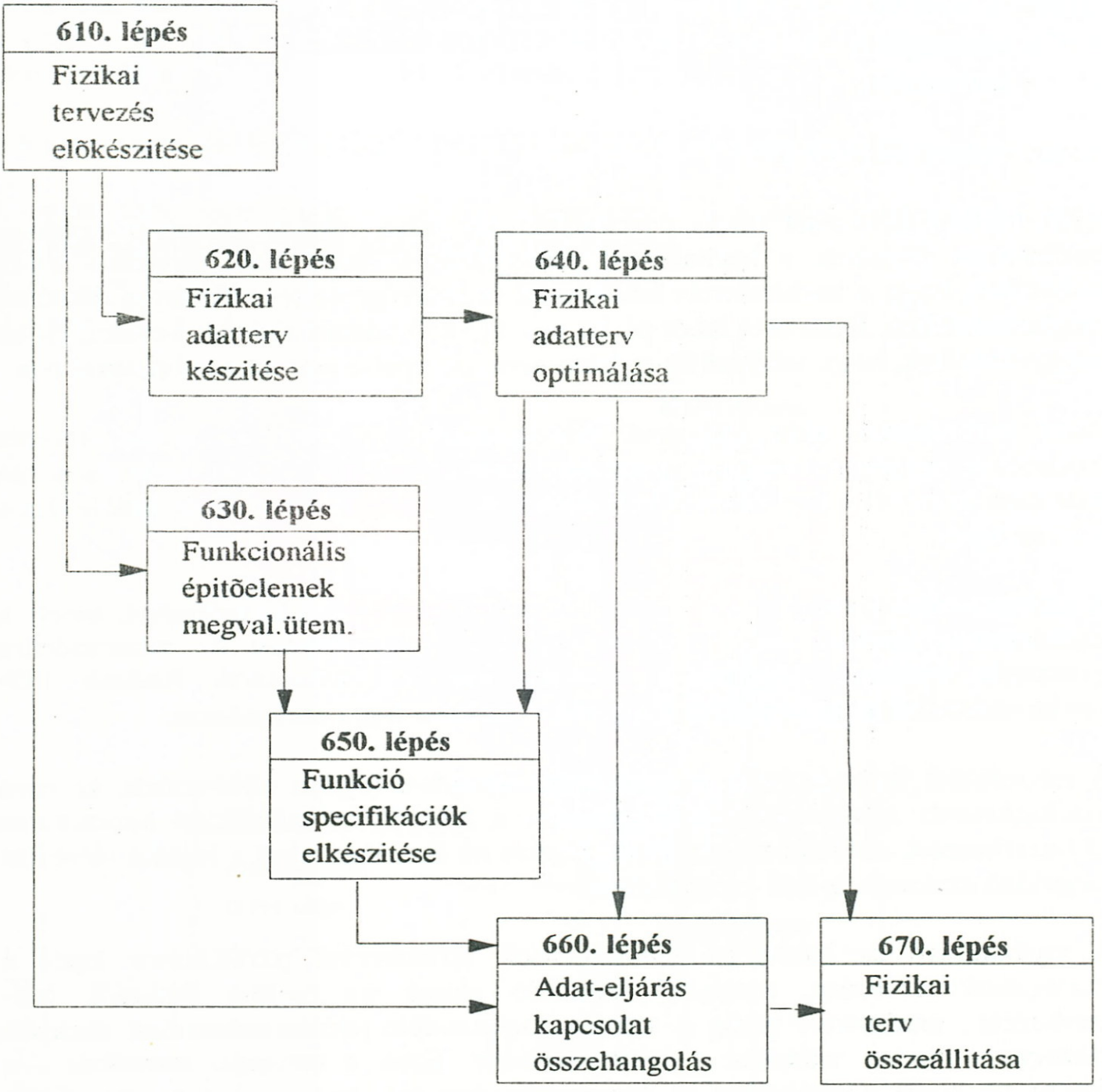
A befejező lépés a minőségellenőrzés végrehajtását, az esetleg szükségessé váló javítások elvégzését, valamint a fizikai rendszerterv-dokumentáció összeállítását jelenti.

A fizikai rendszertervezés fentiekben vázolt lépéseit - egyszerűsített alakban az 10. ábra mutatja be.

A lépések számozása az SSADM-szabvány szerinti, ahol a 6-tal kezdődő számok a 6. szakaszt (fizikai tervezés) jelentik.

6.2 A logikai tervezés

Összehasonlítva a fizikai tervezéssel, ez a szakasz viszonylag egyszerű szerkezetű. Ennek az az oka, hogy az SSADM a tervezési munkát megelőző elemzés során amit csak lehet, elvégeztet a logikai tervezésből is, ezáltal téve lehetővé a rendszertechnikai változatnak a logikai tervezési szakasszal párhuzamos megoldását. Ezért jellemeztem a Követelmény specifikáció szakaszát az előző részben úgy, mint amelyet az elemzés és tervezés közötti átmenetnek is lehet tekinteni.



10. ábra

Logikai szintű adattervezéssel pl. egyáltalán nem kell foglalkozni, mert az adatmodellezés már a korai felmérések során elkezdődik, és mire a logikai tervezési szakasz indul, addigra már készen áll a kívánt rendszer logikai adatmodellje. Az adatmodellek jellegéből fakad ugyanis, hogy eleve logikai szinten készülnek, hiszen ezeknek fizikai szintje nincs is.

A feldolgozások logikai szintű tervezéséhez az SSADM két csoportba sorolja ezeket:

- a lekérdező és
- a karbantartó

feldolgozások csoportjába.

Ezek logikai tervezését részben párhuzamosan lehet végezni, azonban bizonyos elsőbbséget élveznek a karbantartó feldolgozások, mert ezek tervezése során kiderülhet, hogy a karbantartás lekérdezést is szükségessé tesz bizonyos döntések meghozatalához. Ilyen eset lehet pl. ha egy új vevő adatait akarjuk bevinni, előbb meggyőződünk, hogy valóban új-e, tehát nem szerepel-e már az adatbázisunkban.

Akár karbantartó, akár lekérdező feldolgozásról van szó, a logikai tervezés eredménye egységes: *Jackson-struktúra*, amelyet azután a fizikai tervezés során már csak kontroll információval kell ellátni, hogy programterv váljék belőle. Ez a 630-as lépés része.

A karbantartási feldolgozások megtervezéséhez ismerni kell az *eseményt*, amely a karbantartás kiváltó oka, valamint ennek az eseménynek az adatmodellre gyakorolt *hatás*ait. Ugyancsak szükség van a karbantartó funkció I/O-szerkezetére.Ezt az információt a megelőző szakasz fogja szolgáltatni.

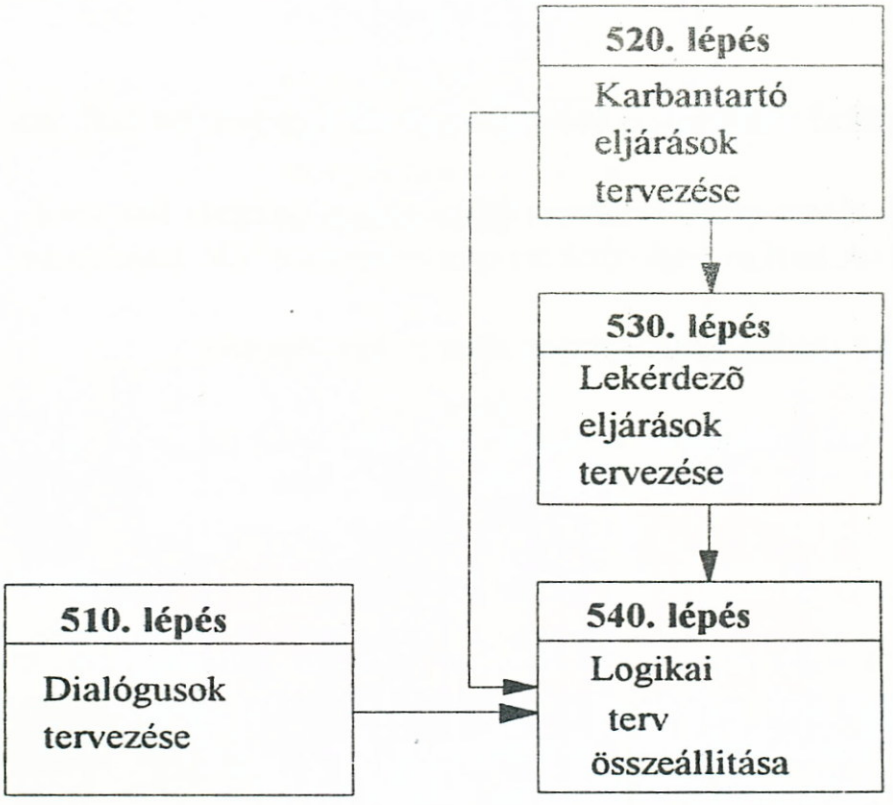
A lekérdezési feldolgozások megtervezése igényli a logikai adatmodell, az ezen meghatározott *lekérdezési utak*, valamint a lekérdező funkciókkal kapcsolatos I/O-szerkezetek rendelkezésre állását. Ezeket az elemeket tehát a logikai tervezést megelőző szakasznak kell szolgáltatnia.

A karbantartó és lekérdező feldolgozások tervezésével párhuzamos lépés a *dialógusok* tervezése, amelynek kiinduló alapja az on-line funkciók I/O-szerkezete., eredménye pedig a szintén Jackson-féle jelölésrendszerben ábrázolt *dialógusszerkezet*, valamint a *menüszerkezet*. Ezek a tervezési termékek - a fizikai tervezési szabályokat is figyelembevéve (pl. hogyan kell a menüknek megjelenni) - kellő alapot képeznek a megfelelő programok elkészítéséhez.

A vázolt gondolatmenetből az is leszűrhető, hogy pontos ismeretekkel kell rendelkeznünk a felhasználói követelményekről. Pl. tudnunk kell, hogy mely lekérdezési funkciókat valósítsunk meg on-line módon, de sok egyéb tényezőt is. Az SSADM megköveteli valamennyi felhasználói követelmény írásban - méghozzá előírt, egységes formában - való rögzítését. Ezt *követelmény katalógus*nak nevezzük és készítése a projekt kezdetekor indul el, majd tartalma és terjedelmefokozatosan bővül mind az öt (vagy 6, a megvalósíthatóság vizsgálata esetén) szakasz során. Vegyük észre, hogy ez a rendszer fejlesztőjének is érdeke, aki így megkímélheti magát a felhasználó utólagos kifogásaitól ("ezt nem így kértük").

A logikai tervezési lépések logikai szerkezetét a 11. ábra mutatja be.

A logikai terv összeállítása - amely az utolsó lépés - ugyanúgy, mint a fizikai terv esetében - a minőségellenőrzést is magában foglalja.



11. ábra

6.3 A rendszertechnikai változat kiválasztása

Az előző, valamint ez a szakasz képezi együttesen a *logikai rendszerspecifikáció*nak nevezett modult.

Kissé paradoxonnak tűnhet, hogy a technikai környezet kiválasztása, amely egyértelműen fizikai tényező, a logikai rendszerspecifikáció része. Nem szabad azonban elfelejteni, hogy *a környezet megválasztása nem tartalmaz fizikai tervezési tevékenységet, csupán annak lehetőségét teremti meg!*

A szakasz lényegében két tevékenységcsoportból áll:

- rendszertechnikai változatok kidolgozása és
- rendszertechnikai változat kiválasztása

A változatok kidolgozásával kapcsolatosan abból kell kiindulnunk, hogy minden logikai terv elvileg többféle műszaki-technikai színvonalon valósítható meg. Ezek természetesen különböző lehetőségeket jelentenek és különböző költségekkel járnak.

A rendszertechnikai változatok kidolgozásánál figyelembe kell venni

- a hosszabb távra szóló rendszerfejlesztési stratégiát (ha van),
- a meglevő technikai eszközökön rendelkezésre álló kapacitást.

A technikai eszközökkel kapcsolatos döntés két lépcsős:

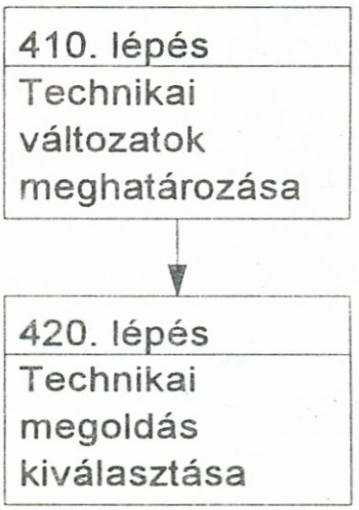
1. azoknak a változatoknak a kiszűrése, amelyek már a nagyvonalú jellemzők alapján láthatóan nem megfelelőek,

2. választás a megmaradt változatok közül azok részletes jellemzése alapján.

Ahhoz, hogy a kiválasztást ilyen módon lehessen végrehajtani, először megfelelő számú változatot kell kidolgozni. Az SSADM által javasolt szám 3-6. Ezeket a változatokat a kapacitásokra, költségekre és a várható hatásaikra vonatkozó néhány globális jellemzővel kell leírni, majd közülük a felhasználó döntése alapján néhányat ki lehet ejteni a további vizsgálatból. A megmaradt változatok száma célszerűen 2-3 legyen. Ezeket azután részletesen le kell írni, hiszen ezek a reálisnak tartott megoldások. A felhasználók a részletes jellemzés alapján döntik el, hogy melyik technikai környezet a megfelelő. A döntéshez a fejlesztőknek igény szerint segítséget kell nyújtani magyarázatokkal, esetleg további részletek kimunkálásával.

A változatok kimunkálásához már szükség van olyan adatokra, amelyek a tárolandó *adatmennyiségek*re, a *feldolgozások*kal és a *felhasználói interfészek*kel kapcsolatos *becslés*ekre lehetőséget adnak. A kiválasztást megelőző szakasznak tehát a logikai adatmodellt el kell látni az adatmennyiségekkel kapcsolatos információval valamint a követelmény katalógust is megfelelő részletességi szintre kell hoznia.

A rendszertechnikai változat kiválasztását végző szakasz lépéseinek logikai szerkezete igen egyszerű, ahogyan az a 12. ábrán látható.



12. ábra

6.4 Összefoglalás

A rendszertervezés három szakaszban történik az SSADM-ben: logikai tervezés, rendszertechnikai változat megválasztása, fizikai tervezés.

A tervezés végeredménye a fizikai terv, amely adatbázis-tervet, programterveket és felhasználói interfész-terveket tartalmaz.

Az ehhez szükséges fontosabb kiinduló információ: logikai adatmodell a várható adatmennyiségekkel és lekérdezési utakkal, a karbantartását kiváltó események és hatásaik leírása, I/O-szerkezetek, funkciómeghatározások és a követelmény katalógus.

7. fejezet
Elemzés SSADM-ben

A logikai tervezés által igényelt input az ezt megelőző elemzési szakaszok outputjaként áll elő. Az elemzéssel az SSADM két modulja foglalkozik. Megvizsgáljuk, hogyan állítják ezek elő a tervezés számára szükséges termékeket, és melyek ezeknek a szakaszoknak a lépései, belső szerkezete. Utalunk a megvalósíthatóság vizsgálatának jelentőségére.

Az előző fejezetben az olvasó képet kapott az SSADM tervezési szakaszairól, valamint arról, hogy melyek azok a legfontosabb termékek, amelyeket a logikai rendszerspecifikáció modulja igényel a megelőző - elemzési - moduloktól, vagyis a *Követelmény elemzés*től és a *Követelmény meghatározás*tól.

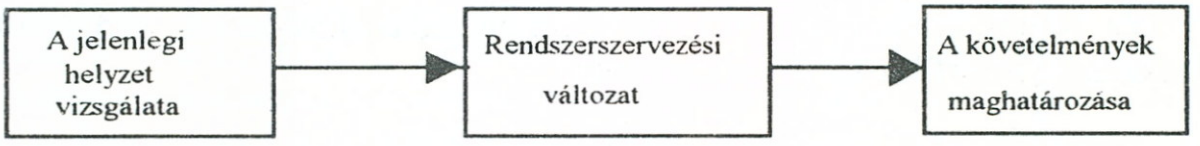
Ez a két elemzési modul - hasonlóan a tervezésnél látottakhoz - ismét három fejlesztési szakaszt jelent, mert a *Követelmény elemzés* két szakaszból áll:

- *A jelenlegi helyzet vizsgálata* és
- *Rendszerszervezési változat kiválasztása*

Ehhez jön a követelmény meghatározás moduljának egyetlen szakasza:

- *A követelmények meghatározása.*

Ugyanúgy, ahogyan ezt a tervezésnél is láttuk, ezek a szakaszok logikai szerkezetet alkotnak az 13. ábrán látható módon.



13. ábra

Ez a szerkezet - eltérően a tervezésnél látottaktól - nem tartalmaz párhuzamosságot. Ennek az az egyszerű oka, hogy csak a jelenlegi helyzet

ismeretében (amibe a követelmények ismeretét is beleértjük) lehetséges rendszerszervezési *megoldási vázlatok*at kidolgozni, ill. azok közül választani. Amikor ez a választás megtörtént, akkor határozzuk meg a követelményeket *részletesen* a választott változatra.

Vizsgáljuk meg a fenti három szakasz tartalmát az eddig is követett módszerünket alkalmazva, vagyis visszafelé haladva.

7.1 A követelmények meghatározása

Ez a szakasz látszólag a legbonyolultabb az SSADM valamennyi szakasza közül a maga nyolc lépésével, amelyekre később visszatérek. A bonyolultság azért látszólagos, mert a korábbiakban megismert fő tevékenység- ill. termékcsoportok itt is nagyon világosan felismerhetők. A csoportosító szempontok:

- *adatok,*
- *funkciók* és
- *felhasználói interfész.*

Összehasonlítva a helyzetet a tervezéssel kapcsolatban megismerttel, itt még nem beszélhetünk adattervekről, csak adatmodellekről, a feldolgozások tervezése helyett a rendszer funkcióit elemezzük (amelyekből a feldolgozástervek levezethetők), az interfészek közül pedig a felhasználóinak a specifikálása a cél.

Tudjuk, hogy adatok vonatkozásában a tervezés számára olyan adatmodellt kell szolgáltatni, amely a várható adatmennyiségekkel kapcsolatos információt is tartalmazza. Tekintettel arra, hogy ez az adatmodell lesz közvetlenül a tervezés alapja, nyilvánvaló, hogy a tervezett, nem pedig a meglevő rendszer adatmodelljéről van szó. Ezt azért érdemes hangsúlyozni, mert a szakmában meglehetősen közismert, hogy a modellezésnél a meglevő rendszer adataiból szoktunk kiindulni. Ez itt is így történik és három lépésben jutunk el a tervezés számára szolgáltatandó modellhez:

- a meglevő rendszer logikai adatmodellje,
- a tervezett rendszer logikai adatmodellje és
- a tervezett rendszer normalizált logikai adatmodellje.

E lépések közül az utóbbi kettő tartozik a követelmények meghatározásának szakaszához, az elsőt a jelenlegi helyzet vizsgálata során végezzük el.

Ahhoz, hogy a tervezett rendszer adatmodelljét a meglevő rendszeréből levezethessük, szükség van a felhasználói követelmények ismeretére, amelyek - mint azt az előző részben láthattuk - az u.n. követelmény katalógusban dokumentáltak. Ezek alapján azt kell elemezni, hogy milyen részekkel kívánatos kiegészíteni a meglevő rendszer modelljét (egyed-, tulajdonság- és kapcsolattípusok), vagy esetleg mit lehet elhagyni a modellből, mert feleslegessé válik az új rendszerben.

Itt érdemes kitérni arra, hogy az adatmodellezés az SSADM-ben először az u.n. logikai adatszerkezet kialakítását jelenti, amely nem más, mint egy *egyed-kapcsolati modell*, igen kevés tulajdonságtípussal (elsősorban kulcsokkal) felszerelve. Ez az indítás megfelel annak, hogy a fejlesztés kezdeti szakaszaiban még kevesebb ismeretünk van a részletekről. Amikor aztán az adatelemzéssel párhuzamosan folyó folyamatelemzés eljut a funkciók input/output követelményeinek meghatározásához, akkor lehet elvégezni a *relációs adatelemzés*t (normalizálást) az inputokat, ill. outputokat, mint tulajdonságtípus-forrásokat felhasználva.

Ez a relációs adatelemzés független a korábban kialakított egyed-kapcsolati adatmodelltől. Úgy is mondhatjuk, hogy az egyedkapcsolati modellt felülről lefelé haladó (top-down) módon a rendszer egészére vonatkozó ismeretek (interjúk, dokumentumok) alapján határozzuk meg, míg a normalizált modellt a tulajdonságtípusokból kiindulva, tehát alulról felfelé haladva (bottom-up) fejlesztjük ki, majd a kétféle eredményt összehasonlítva és egymással ellenőrizve határozzuk meg a végleges modellt.

A funkciók elemzését, ill. meghatározását az adatelemzéshez hasonló fokozatosság jellemzi. Azok a funkciók, amelyek I/O-szerkezete képezi az alapját a relációs adatelemzésnek - valamint más további tervezési lépéseknek - a következő lépésekben alakulnak ki:

- meglevő rendszer fizikai folyamatai,
- meglevő rendszer logikai folyamatai,
- tervezett rendszer logikai folyamatai,
- tervezett rendszer funkciói.

A követelmény meghatározás - amint arról korábban már szó esett - arra a megoldási változatra készül el, amelyre a felhasználói vezetés választása esett. Amíg a választás nem történik meg, addig nincs értelme tervezett rendszerről beszélni, következésképpen a fenti lépések közül csak a második kettő tartozik a követelmények meghatározásához.

A folyamatok elemzése az u.n. *adatfolyam modellezés*i technika[3] segítségével történik. Ennek végeredményéből lehet a funkciókat meghatározni, egymáshoz rendelve az összetartozó feldolgozásokat, inputokat, outputokat, valamint adatmodellbeli adatokat. A funkciók ilyen értelmű meghatározottsága, valamint a végleges adatmodell rendelkezésre állása teszi lehetővé az u.n. *feldolgozás-specifikációk* elkészítését. Ez alatt a tervezés által igényelt olyan termékeket értünk, mint a lekérdezési utak meghatározása, továbbá a karbantartást kiváltó események és hatásaik leírása. Az utóbbi tevékenység technikáját *eseményhatás*[4] *modellezés*nek nevezik

Az *I/O-szerkezetek* meghatározása (a funkció meghatározás részeként) válik az alapjává a fentebb megemlített hamadik nagyobb tevékenységcsoportnak, a felhasználói interfész meghatározásának. Itt csak emlékeztetni szeretnék arra, hogy a feldolgozás-adat interfész meghatározása a fizikai tervezés része (ld. az előző fejezetben).

A felhasználói interfész meghatározásához az SSADM a specifikáció *prototipizálás* technikáját ajánlja (ellentétben pl.az u.n. iteratív fejlesztéssel). Másszóval a prototípus szerepe midössze annyi, hogy segít pontosan tisztázni a felhasználóval a felhasználói felülettel szembeni pontos követelményeket. Ennek a lépésnek az eredményeként alakulnak ki a képernyőtervek, a dialógusok és menük igénye, amit - mint láttuk - a tervezés használ kiindulópontként.

A munkaszakasz utolsó előtti lépése fontos minőségellenőrzési mérföldkő: az eredetileg kitűzött célok ellenőrzése érvényesség szempontjából. E szakasz végén nagyjából a projekt közepén járunk. Már nagyon sok információval rendelkezünk, amelyek esetleg más megvilágításba helyezik az egész rendszert! Ugyanakkor itt még viszonylag kis veszteséggel tudunk változtatni a célkitűzéseken, esetleg le is állíthatjuk a projektet.

Végül a szakasz dokumentációjának az összeállítása következik, ami a már a munka menetében elkészült dokumentumok megfelelő rendszerbe foglalását, szerkesztését és a kísérő szöveg megírását jelenti. A követelmény meghatározás szakaszának logikai szerkezetét a 14. ábra mutatja be.

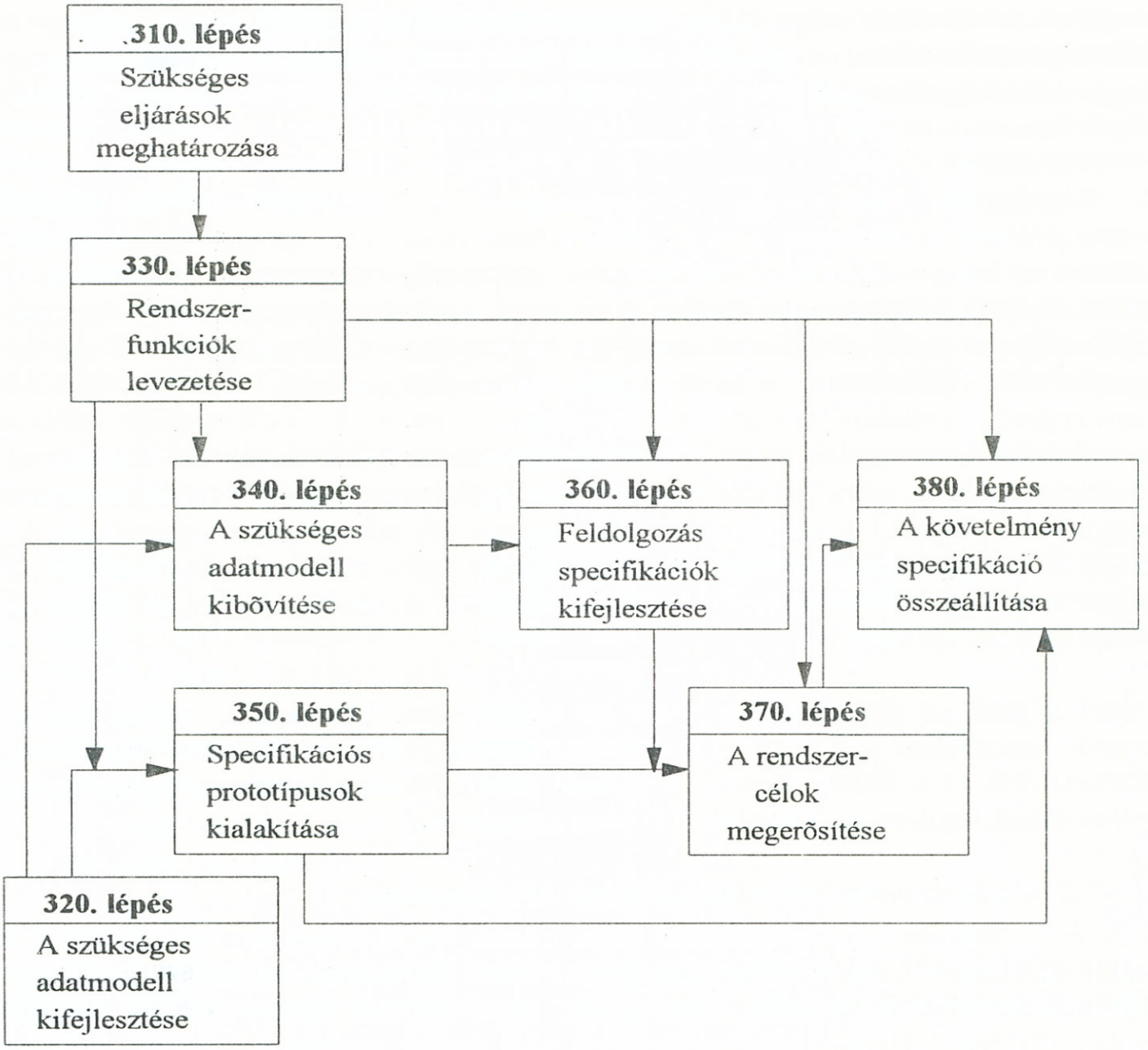
7.2 A rendszerszervezési változat kiválasztása

A tervezésnél már találkoztunk egy kiválasztási szakasszal, a rendszertechnikai változat kiválasztásával. Látni fogjuk, hogy az elvégzendő lépések jellegét illetően

[3]Leírását ld. a 3. Részben.
[4]Leírását ld. ugyanott.

nagyfokú a hasonlóság, van azonban - többek között - egy nagyon lényeges különbség: A rendszertechnikai változat esetében azt a *környezetet* kellett megválasztani, amelyre a fizikai terv készült, itt viszont egy rendszerszervezési *megoldást* kell kiválasztani.

Ennek a választásnak az alapgondolata az, hogy ne erőszakoljunk rá a felhasználóra egy meghatározott szervezési elképzelést, még akkor sem, ha nekünk - mint rendszerszervezőknek - van egy dédelgetett megoldásunk. Kínáljunk fel néhány változatot, megmutatva mindegyiknek a legfontosabb jellemzőit, aztán hadd döntse el a felhasználó, hogy neki melyik a rokonszenves.



14. ábra

A rendszerszervezési változatokkal kapcsolatos döntés két lépcsős:

1. azoknak a változatoknak a kiszűrése, amelyek már a nagyvonalú jellemzők alapján láthatóan nem megfelelőek,
2. választás a megmaradt változatok közül azok részletes jellemzése alapján.

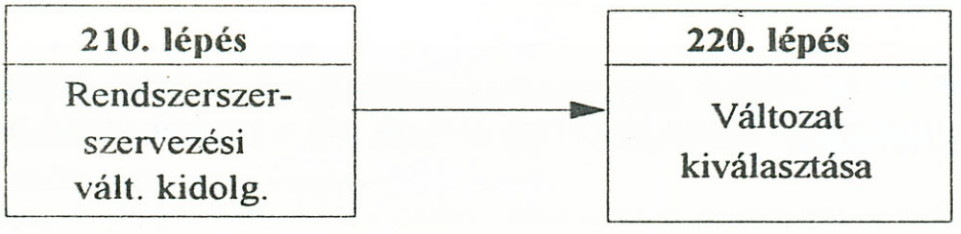
Ahhoz, hogy a kiválasztást ilyen módon lehessen végrehajtani, először megfelelő számú változatot kell kidolgozni. Az SSADM által javasolt szám 3-6. Ezeket a változatokat a költségekre és a várható hatásaikra vonatkozó néhány globális jellemzővel kell leírni. Ilyen jellemzők: a rendszer határai, inputjai, outputjai és annak vázlatos leírása, hogy mit csinál a javasolt megoldás, majd közülük a felhasználó döntése alapján néhányat ki lehet ejteni a további vizsgálatból.

A megmaradt változatok száma célszerűen 2-3 legyen. Ezeket azután részletesen le kell írni, beleértve várható költségeiket, a becsült előnyöket és megtakarításokat, valamint a szervezeti kihatásokat,hiszen ezek a reálisnak tartott megoldások. A felhasználók a részletes jellemzés alapján döntik el, hogy melyik megoldási változat a megfelelő. A döntéshez a fejlesztőknek igény szerint segítséget kell nyújtani magyarázatokkal, esetleg további részletek kimunkálásával.

Ezeknek a változatoknak a leírása természetesen logikai szintű, de nem túl részletes. Az SSADM kézikönyvek pontosan közlik a részletesség javasolt szintjét. Ennél részletesebb specifikációt egyszerűen nem érdemes készíteni, hiszen még nem tudhatjuk, mi lesz a felhasználó döntése. Csak az ehhez a döntéshez feltétlenül szükséges mélységet célozzuk meg, nehogy túl sok munka vesszen kárba a döntés eredményeként!

Gyakran előfordul, hogy a választás nem egyszerűen valamelyik felkínált változatra esik, hanem módosításokra van szükség, esetleg több javasolt változat kombinációját kívánja a felhasználó.

Ennek a szakasznak a belső felépítését látjuk a 15. ábrán. (A számozásban a 2-es utal arra, hogy ezek a 2. szakasz lépései.)



15. ábra

Ahhoz, hogy el lehessen készíteni rendszerszervezési változatokat, meg kell történnie a meglevő helyzet felmérésének.

7.3 A jelenlegi helyzet vizsgálata

Ez az SSADM u.n. magját képező hat fejlesztési szakasz közül az 1-es sorszámú. Célja, hogy alapot teremtsen a rendszerszervezési változatok kimunkálásához.

Az alapgondolat az, hogy az új rendszert akkor tudjuk igazából létrehozni, ha pontosan megértjük, hogyan kell majd működnie. Az esetek csekély százalékától eltekintve, valamilyen létező információs rendszert, vagy szoftvert kell az újnak felváltania, amelynek a megismerése ad módot mind a benne rejlő hibák feltárására, mind pedig az új rendszertől elvárt többlet-szolgáltatások megértésére.

Ez a megismerési folyamat is bizonyos fokozatokban történik. Először - a dolog természetéből adódóan - a meglevő (vagy a kívánthoz igen hasonló), fizikailag létező rendszert tanulmányozzuk. Ilyenkor tehát legtöbbször fizikai szintű rendszer ismeretre teszünk szert. Ez persze magában hordozza mindazokat a kötöttségeket, amelyeket ennek a rendszernek a létrehozásakor kellett figyelembe venni, ili. amelyek idők folyamán mint módosítgatások, vagy szükségmegoldások rakódtak rá. Ezek a működés belső lényegét jórészt elfedik, ezért meg kell tőlük szabadulni.

A második fokozatban kiszűrjük, eltávolítjuk a fizikai elemeket és így jutunk el a meglevő rendszer logikájának a megismeréséig. Itt ér véget a megismerésnek az a része, amely ebben a fejlesztési szakaszban kell, hogy megtörténjen. A következő fokozat, a kívánt (tervezett) új rendszer logikájának a kialakítása a későbbi szakaszok feladata - amint azt az olvasó már megismerte.

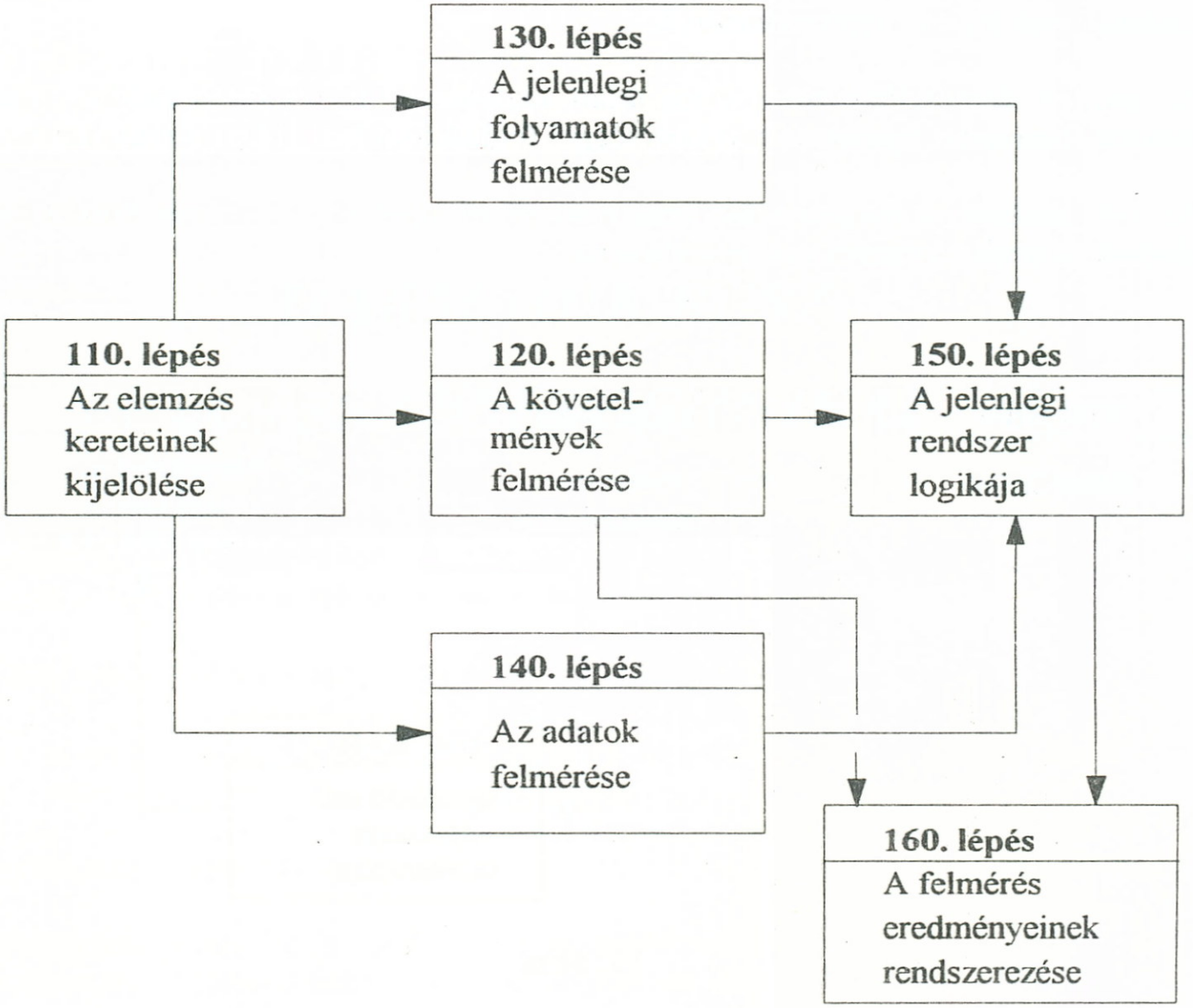
Ebben a felmérési jellegű szakaszban a meglevő rendszer megismerésének három fő irányát különböztetjük meg:

- *a folyamatok feltárása,*
- *az adatok feltárása* és
- *a követelmények meghatározása.*

A folyamatok és az adatok modellezése - amint azt fentebb kifejtettem - fizikai szinten indul, majd egy külön lépésben állítjuk elő a logikai szintű modelleket.

Megelőzi ezeket a lépéseket a szakaszt indító előkészítő lépés, a szakasz munkáját pedig - szokásos módon - a dokumentáció összeállítása zárja.

A jelenlegi helyzet vizsgálatát alkotó lépések logikáját a 16. ábra mutatja be.



16. ábra

7.4 A megvalósíthatóság elemzése

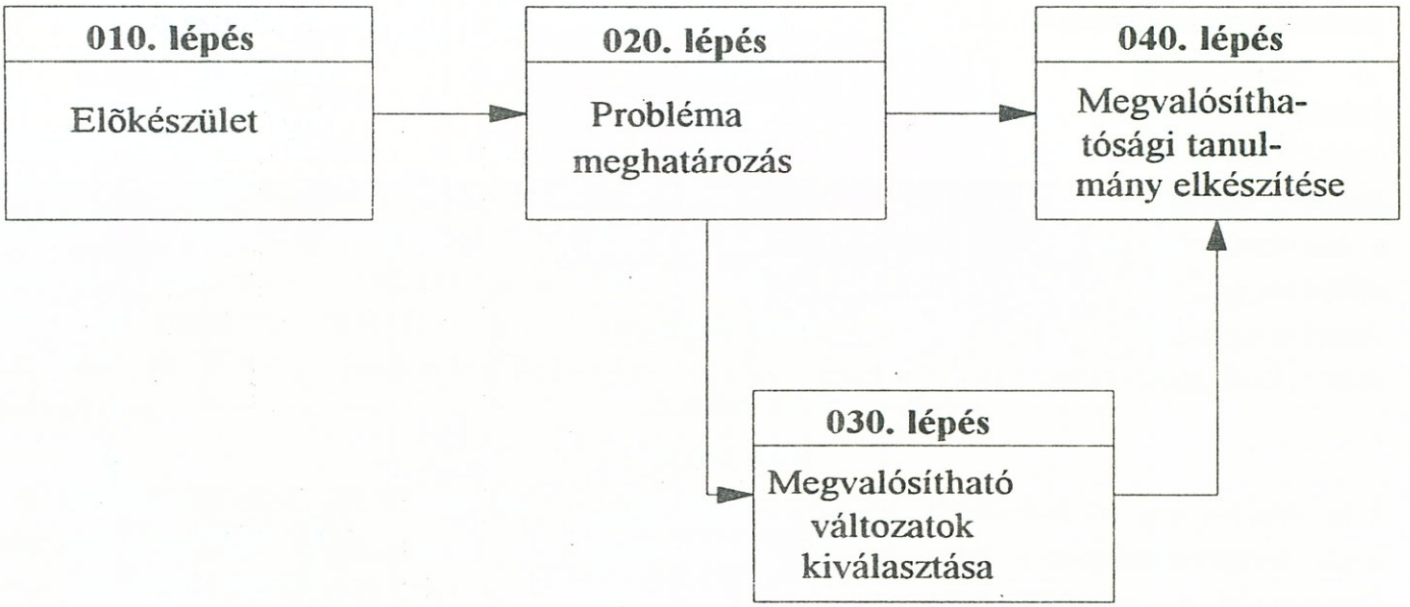
Ez az SSADM 0. szakasza, amely egyben az öt modul közül az első. Két fontos dolgot kell róla tudni:

- nem tartozik a kötelezően végrehajtandó szakaszok közé,
- ha alkalmazzuk, akkor az a célja, hogy bemutassa, érdemes-e egyáltalán elindítani a szóbanforgó projektet.

Mivel nem kötelező, ezért kihagyása esetén a követelmény elemzési modul veszi át a szerepét. Ha viszont mégis elvégezzük, akkor a követelmény elemzés szakaszainak lépései egyszerűsödnek.

Az említett döntéselőkészítési jelleg miatt vázlatosan, ill. rövidített módon szinte minden fontosabb fejlesztési tevékenységgel foglalkozni kell ebben a szakaszban.

Az 1-6. szakoszok megismerése után nem jelent gondot a 17. ábrán bemutatott lépések és azok szerkezetének értelmezése.



17. ábra

HARMADIK RÉSZ
AZ SSADM TECHNIKÁI

8. fejezet
A technikákról általában

Az előző részben volt szó arról, hogy a technikák írják le, hogyan kell végrehajtani az SSADM szerkezetében megadott *feladatokat*. Ezek azok a módszerek, amelyek a szerkezeti keretben szervesen egymásra épülve eredményezik a módszertant, beleértve az előállított termékek specifikációit is.

Az SSADM hozzávetőlegesen egy tucatnyi tachnikát foglal magába. Talán meglepő a "hozzávetőlegesen" kifejezés használata egy olyan módszertannal kapcsolatban, amelyről azt állítottam, hogy igen pontosan meghatározott. A bizonytalanság valóban csak látszólagos, mert nem a módszertanban van, hanem a róla alkotott véleményekben. Vannak ugyanis olyan módszerek, amelyeket nem minden szakember sorol az önálló technika kategóriájába.

Elöljáróban két nagy csoportra osztjuk a technikákat:

- a diagramra épülő és a
- nem diagramszerű

módszerek csoportjára.

Abból kiindulva, hogy az SSADM alapvető célkitűzése a rendszerfejlesztéshez szükséges kommunikáció támogatása, nagyon sok esetban használja ki a képi ábrázolás lehetőségét. Sok tehát az olyan technika, amely diagramot, vagy diagramokat használ. Ezek a következők:

- logikai adatmodellezés,
- adatfolyam modellezés,
- egyed-esemény modellezés,
- elérési út modellezése,
- I/O szerkezet meghatározása,
- dialógus tervezés,
- feldolgozások logikai modellezése.

Azért mondjuk ezekre, hogy diagramra *épülnek*, mert nem egyszerűen diagramokról van szó. A diagramokhoz tartozik egy sereg információ, ami nem is adható meg magán a diagramon, tehát minden diagramhoz többé-kevésbé

terjedelmes háttérinformáció tartozik. Ennek megjelenési formája attól függ, hogy a dokumentáció kézzel készül-e, vagy számítógéppel. A kézi dokumentációhoz formanyomtatványok szükségesek, amelyek közül a legalapvetőbbek már magyar fordításban is rendelkezésre állnak.[5]

A nem diagramszerű technikák:

- követelmény meghatározás
- rendszerszervezési változatok kidolgozási módszere,
- rendszertechnikai változatok kidolgozási módszere,
- funkció meghatározás,
- relációs adatelemzés,
- specifikáció prototipizálás,
- fizikai tervezés.

Az egyes technikákkal kapcsolatban gyakran használt kifejezés a *specifikáció*, ill. a *dokumentáció*. Érdemes röviden átgondolni, mi a különbség a két fogalom között.

A specifikáció olyan követelmények pontos megadása, amelyeket egy berendezésnek, vagy valamely munka végtermékének teljesítenie kell. Úgy is mondhatnánk, hogy *a specifikáció előírás*.

Ezzel szemben *a dokumentáció bizonyos tények utólagos rögzítését jelenti*, tehát egy, már elkészült állapotot tükröz vissza.

Olyan esetekben, amikor többlépcsős munkáról van szó - mint pl. információs rendszerek fejlesztésekor - az egyes szakaszok végtermékének dokumentációja egyben rögzíti a következő szakasszal szembeni követelményeket, vagy azok jelentős részét, tehát ilyen értelemben specifikációként is felfogható. Mindenesetre célszerű az adott helyzet függvényében megválasztani a helyes kifejezést, ezzel is csökkentve a félreértés lehetőségét.

[5] SSADM Struktúrált rendszerelemzési és -tervezési módszer; MTA Információtechnológiai Alapítvány, Budapest, 1993.

9. fejezet
Folyamatmodellezés

9.1 A folyamatmodellezés helye és szerepe

A folyamatmodellezés célja, hogy segítségével a rendszerszervező le tudja írni az információs rendszereket. Nemcsak az azokban zajló folyamatokat - amit a név alapján feltételezni lehetne - hanem az adatok tárolását, mozgását, valamint forrásait és/vagy felhasználóit.

Lássuk, melyek azok az esetek a rendszerfejlesztés során, amikor ilyen értelmű leírásra van szükség:

- Felméréskor (SSADM 1. szakasz) a meglevő rendszer leírását kell elkészíteni, aminek az a célja, hogy ellenőrizni lehessen az alapkövetelmények helyes megértését. Az elemző szakember (csoport) általában több változaton keresztül éri el a rendszer helyes leírását. Az egyes változatokat megvitatja a felhasználókkal és fokozatosan javítja a leírást.

- Új rendszer tervezésekor a szervezőnek le kell írnia a rendszer jövőbeli működését, hogy megvitathassa azt a felhasználókkal és elnyerhesse jóváhagyásukat a megvalósításhoz.

- Amikor a felhasználónak választania kell a különböző lehetséges megoldások közül, a szervező egy sor rendszerleírást készít ezekről a megoldásokról, elemezve a hozzájuk kapcsolódó várható költségeket és hozamokat is.

A fent felsorolt esetekben a rendszerleírás folyamatosan módosul a felhasználókkal folytatott megbeszélések és a tervezés részletesebbé válásának eredményeként. Ennek alapján meg is fogalmazhatunk bizonyos követelményeket a leírás módszerére vonatkozóan:

- legyen könnyen és gyorsan előállítható,

- legyen könnyen és gyorsan változtatható,

- legyen könnyen érthető mind a számítástechnikai szakemberek, mind a számítástechnikában nem járatos felhasználók számára,

- tegye lehetővé a mindenkori követelményeknek megfelelő részletezettséget, legyen az akár kevés, akár sok,

- feleljen meg annak az elvnek, mely szerint a fejlesztésben viszonylagos elsőbbséget kell biztosítani az adatoknak.

A korábban rendszerleírás céljára rendelkezésre álló eszközök a közönséges szöveges leírások valamint a folyamatábrák voltak. Ez utóbbiak közé soroltuk a program- és rendszer-folyamatábrákat - hogy csak a legismertebbeket említsem.

Vajon kielégíti-e a szöveges leírás, vagy a hagyományos folyamatábrák technikája ezeket a követelményeket?

18. ábra

A szöveges leírások főbb hátrányai a következők:

- nehezen szabványosíthatók, stílusuk, érthetőségük nagy mértékben függ az adott személy fogalmazási képességétől;

- vagy a pontos leírásra való törekvés, vagy az egyéni stílus következtében igen könnyen válnak hosszúvá, s így elkészítésük időigényes;

- a szöveg íróját semmi sem készteti automatikusan az adatközpontú közelítésmód alkalmazására;

- ha nem kimondottan tehetségesen fogalmazó szakember írja, akkor az olvasó könnyen elveszhet a részletekben anélkül, hogy megértené a lényeget.

A hagyományos folyamatábráknak ugyancsak vannak olyan tulajdonságaik, amelyek rendszerleírási szempontból nem előnyösek:

- eljárás-orientáltak, tehát elsősorban az eljárásokra összpontosítanak, az adatok csak mint az eljárások szükségletei jelennek meg (a rendszerszervezés korai korszakának technikája; ld. 1. Rész);

- keverednek bennük a rendszerek logikai és fizikai vonatkozásai, tehát nem választják el világosan a "mit old meg a rendszer?" kérdést a "hogyan oldja meg a rendszer?" kérdéstől;

- nem teszik lehetővé a könnyű átmenetet az áttekintő leírások szintjéről a részletek bemutatásához;

- nem-számítástechnikai szakemberek számára nehezen érthetők;

- elkészítésük és módosításuk időigényes;

- a sorrendiség jelölésén kívül az egyes blokkjaikat összekötő nyilaknak nincsen semmilyen tartalmuk.

Láttuk, hogy a strukturált módszertanokban milyen fontos szerepe van a fejlesztők és felhasználók közötti kommunikációnak, egymás világos megértésének. Ebből a szempontból az ábrák (diagramok) használatának nagy a jelentősége, tehát a rendszerleírás alapeszköze továbbra is a diagram kellett, hogy maradjon, de a korábbinál használhatóbb formában.

Az SSADM-ben a folyamatmodellezés eszköze az u.n. adatfolyam diagram (adatáramlási diagram), valamint a hozzá kapcsolódó kiegészítő dokumentáció. Az adatfolyam diagramok (AFD) használatával a fenti hátrányok kiküszöbölődnek, ill. minimálisra csökkennek. A rendszerleírásokkal kapcsolatban megfogalmazott követelményeket ezek a diagramok a következőképpen teljesítik:

- gyakorlatiasak, nem technikai jellegűek, ezért gyorsan elkészíthetők és módosíthatók;

- könnyen érthetők a felhasználó számára is, tehát hatásosan támogatják a rendszer fejlesztői és felhasználói között szükséges kommunikációt;

- szintekre bontott, hierarchikus szerkezetben készíthetők, s ezáltal támogatják a felülről lefelé haladó szemléletet;

- minimális átfedéssel (redundanciával) készíthetők, és így megkönnyítik a változások átvezetését;

- segítségükkel a rendszerek akár fizikai, akár logikai szempontból leírhatók és a fizikai AFD-k logikaivá alakíthatók.

Természetesen megvannak itt is a betartandó szabályok, akárcsak a hagyományos folyamatábrák rajzolásánál, de megkönnyíti a dolgot, hogy az AFD valamennyi elemének megvan a konkrét, valóságbeli megfelelője, beleértve a nyilakat is.

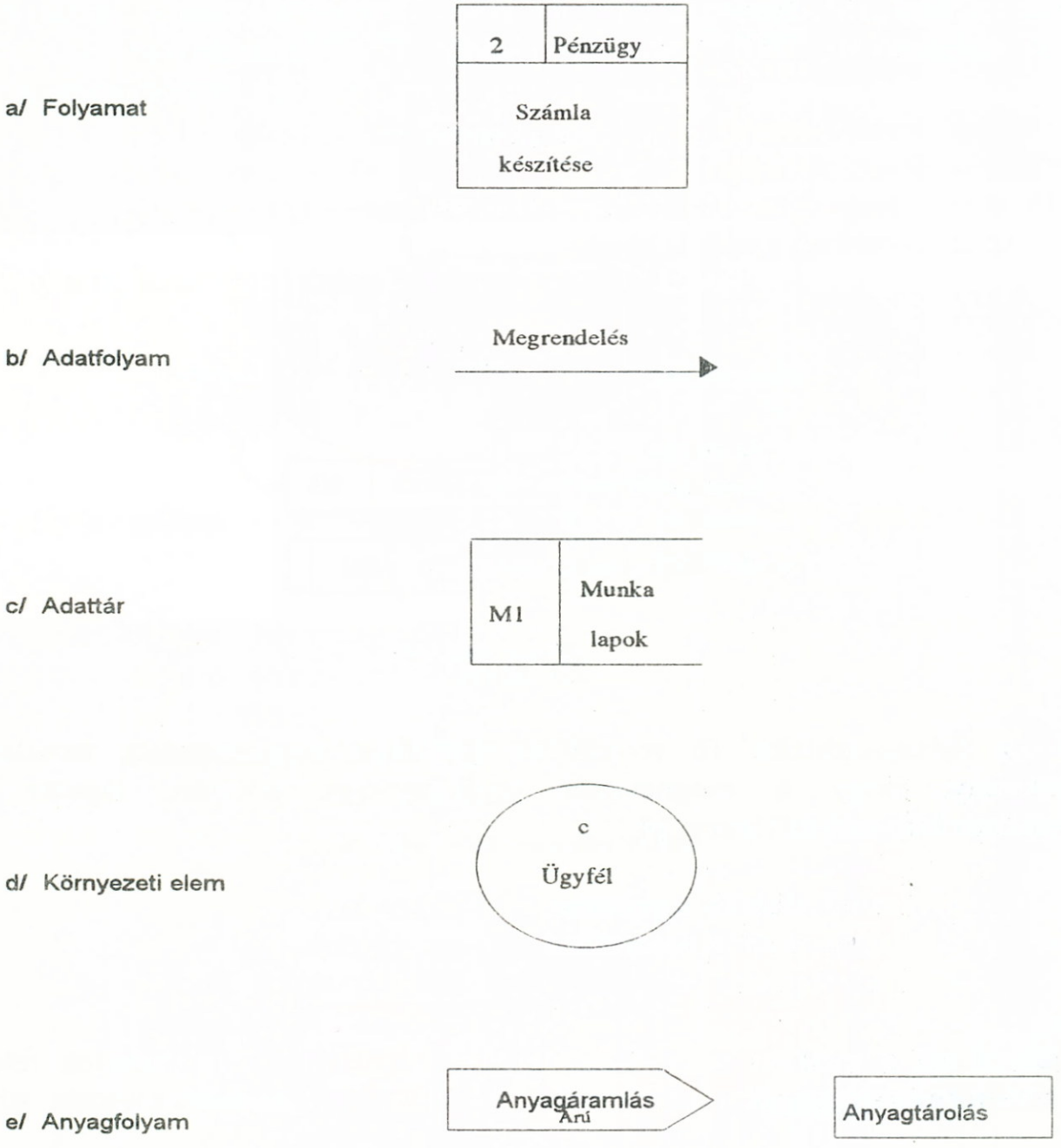
Az adatfolyam diagramokat - előnyös tulajdonságaik miatt - az elemzés és tervezés során kiterjedten használjuk. A meglevő rendszer vizsgálatakor annak fizikai szintű AFD-it készítjük el, majd ezeket logikaivá alakítjuk. A követelmény specifikáció szakaszában azután a meglevő rendszer logikáját a kívánt rendszer logikájává dolgozzuk át, miután a lehetséges megoldások vázlatait (szintén logikai AFD-k) a felhasználó kiértékelte és választott közülük.

9.2 Az adatfolyam diagram elemei

Az AFD-n öt különböző elem fordulhat elő, amelyeket a 19. ábrán látunk. Egy-egy konkrét AFD nem feltétlenül tartalmazza mind az öt itt felsorolt elemet.

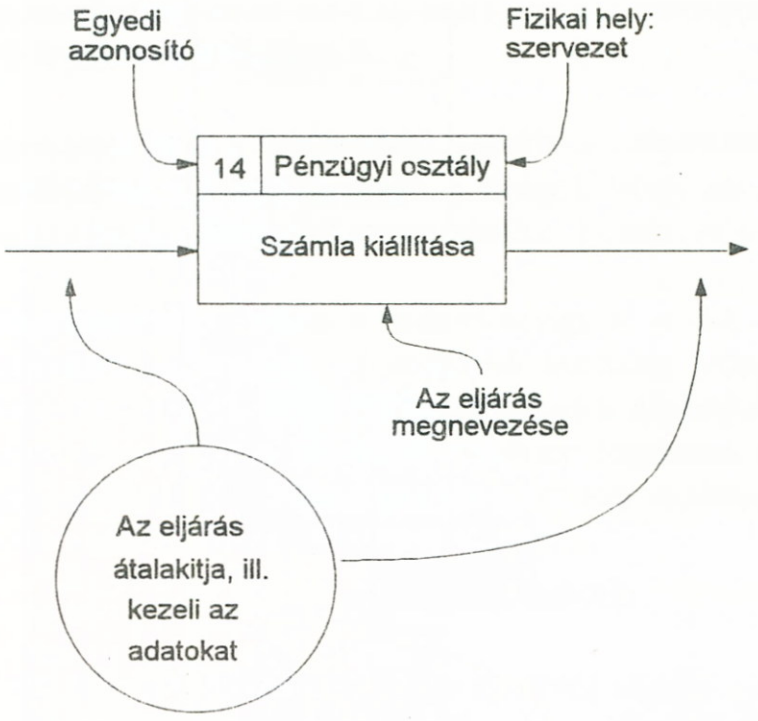
Folyamat

Téglalap jelképezi az eljárásokat, tehát a rendszer azon tevékenységeit, amelyek az információ-átalakítást végzik. A téglalap három részre van felosztva. A bal felső sarokba írjuk az azonosítót, amely egyszerű sorszám. Általában a diagramon felülről lefelé, és/vagy balról jobbra haladva önkényesen rendeljük a folyamatokhoz. Fontos, hogy ezek a számok *nem jelölnek semmiféle sorrendiséget*, pusztán az azonosítás célját szolgálják.



19. ábra

Az azonosító melletti rovatba annak a szervezeti egységnek a nevét írjuk, amely felelős az adott folyamat végrehajtásáért. Esetleg egy munkahelyi beosztás neve is kerülhet ide (pl. raktáros). Az ún. logikai AFD-ken ez a rovat üresen marad.



20. ábra

Végül az folyamatblokk fő rovata az folyamat megnevezését tartalmazza. Követelmény, hogy a megnevezés tevékenységet jelöljön, legyen minél pontosabban kifejező és tömör, pl.:

- munkafelelős kijelölése,
- megrendelés besorolása,
- helyfoglalás, stb.

Kerüljük el azt a gyakori hibát, hogy gyorsan átsiklunk a folyamatok felett, és olyan semmitmondó neveket adunk nekik, mint feldolgozás, karbantartás, stb.

Adatfolyam

Az AFD-n látható nyilakat leginkább olyan csatornaként képzelhetjük el, amelyeken keresztül adatok áramolnak a diagram többi elemei között. Ezért nevezik egyes szakkönyvek adat-csővezetéknek is (data pipeline).

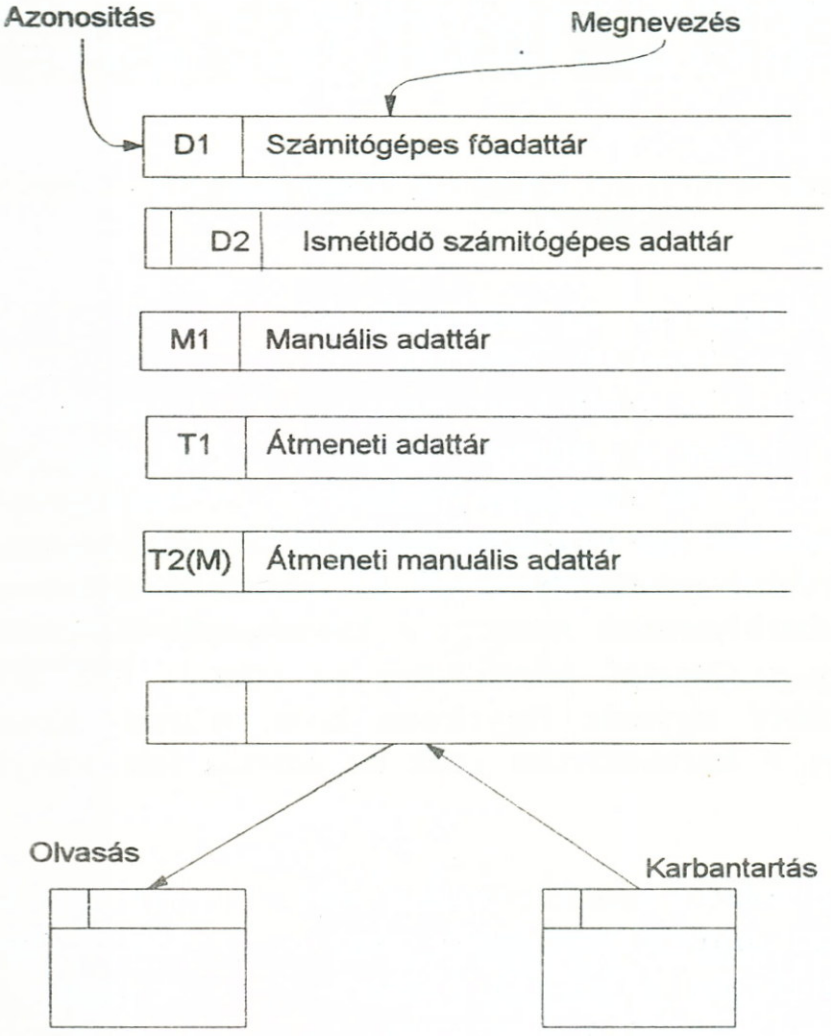
A nyíl hegye jelöli az adatáramlás irányát. Maga a nyíl lehet akár egyenes, akár görbült.

Az adatfolyamoknak — lehetőleg egyedi — nevet kell adni, amelyet a nyílra írunk. Természetesen a többi megnevezéshez hasonlóan, ennek a névnek is kifejezőnek kell lennie. Egyes esetekben (pl. fájl karbantartásakor), az adatfolyam

tartalma eleve nyilvánvaló, és ilyenkor a név el is hagyható. Meglevő rendszerek leírásakor a használatban levő dokumentumok nevét alkalmazhatjuk, az adatfolyamok többsége ugyanis ezeket jelképezi.

Adattár

Jobbról nyitott, keskeny téglalap jelképezi az adattárakat.



21. ábra

Szándékosan nem nevezzük ezeket sem fájlnak, sem állománynak, a cél ugyanis az, hogy a szimbólum minél általánosabb értelmű legyen. Jelképezhet manuális tárolást, mint pl. kartoték fiókot, számítógépes fájlt, vagy akárcsak egy irattartó tálcát az íróasztalon, amelyben a bejövő, vagy kimenő leveleket, dokumentumokat átmenetileg tárolják.

Az adattárak kétféle célt szolgálhatnak egy információs rendszerben:

- Vannak olyan adatok, amelyekre gyakran van szükség különféle információk előállításához, de költséges, időrabló lenne, sőt állandó hibaforrást jelentene az újra, meg újra történő bevitelük. Ezek azok az adatok, amelyeket törzsadattárakban szoktak tárolni;

- Előfordul, hogy olyan adatok jelennek meg a rendszerben (pl. valamelyik eljárás kimeneteként), amelyekre nem azonnal, hanem valamikor később lesz szüksége valamelyik másik eljárásnak. Az ilyen adatok átmeneti adattárakba kerülnek.

Valamennyi adattár a fenti két kategória valamelyikébe — esetleg mind a kettőbe — tartozik.

Az adatfolyam diagramon az adattárak felé irányuló, vagy azokból kilépő adatfolyamok kétféle típusúak lehetnek. Ha az adattár csupán visszakeresési célokat szolgál, akkor az adatfolyam egyirányú, az adattárból halad a megfelelő eljárás felé. Ha viszont az adattár karbantartásáról van szó, akkor előbb a megfelelő rekordot, — ill. manuális rendszerben dokumentumot — előbb ki kell keresni, módosítani kell egy eljárással a megfelelő adatokat, majd a karbantartott rekord (dokumentum) visszahelyezendő az adattárba. Ha most gondolatban a kétféle irányú adatfolyamnak mintegy a különbségét képezzük, akkor könnyen rájöhetünk, hogy a "tiszta" adatfolyam az adattár felé áramló karbantartó információ, a többi ugyanis figyelmen kívül marad. Emiatt tehetjük meg legtöbbször, hogy a karbantartást csak az adattár felé irányló adatfolyammal jelöljük.

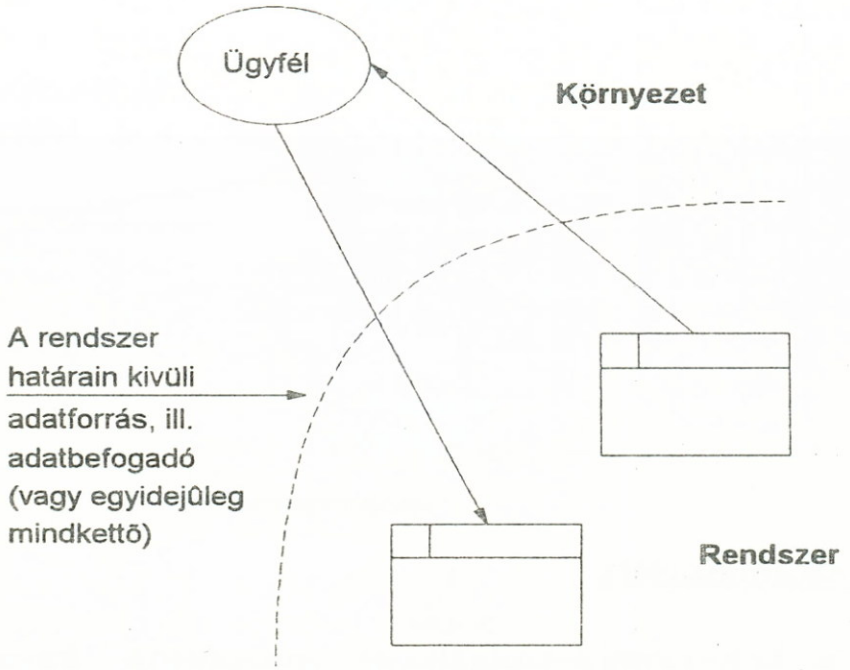
Az adattárral kapcsolatos adatfolyamokat sokszor névvel sem kell ellátni. Ha pl. egy megrendelésen szereplő vevő nevét, vagy egyéb adatait ellenőrző eljárás felé irányul egy adatfolyam a VEVŐ adattárból, akkor ennek tartalma magától értetődő. Ez a megállapítás a diagramra vonatkozik, egyébként az adatfolyam pontos leírását külön dokumentum tartalmazza.

Az adattárakat is azonosítóval látjuk el, amely betűből és számból áll. Meglevő, vagy megvalósítandó rendszerek fizikai részleteit leíró diagramokon "M" betű jelöli a nem-számítógépi (manuális) adattárat, "D" pedig a számítógépeset. Az ú.n. logikai AFD-ken az adattárak azonosító rovatában "L" betűt is használhatunk, sorszámmal kombinálva.

Ugyanaz az adattár több különböző helyen is berajzolható az AFD-n, ha így el tudjuk kerülni az adatfolyamok nyilainak kuszaságát. Ilyenkor a többszörözésre azzal hívjuk fel a figyelmet, hogy az adattár szimbólumának baloldali határoló vonalát megkettőzzük (21. ábra).

Környezeti elem:

A vizsgált rendszer határain kívül eső adatforrást, vagy adatok címzettjét jelöli. Azért lényeges ezeknek az ábrázolása, mert ugyan a rendszer határain kívül vannak, annak működését azonban befolyásolják. Környezeti elem lehet pl. partnervállalat, vagy más belső szervezeti egység, esetleg hivatal (pl. KSH), stb.



22. ábra

Szimbóluma ellipszis alakú, esetleg - megfelelő sablon hiányában - a program-folyamatábrák határoló (START/END) szimbólumával egyezik meg.

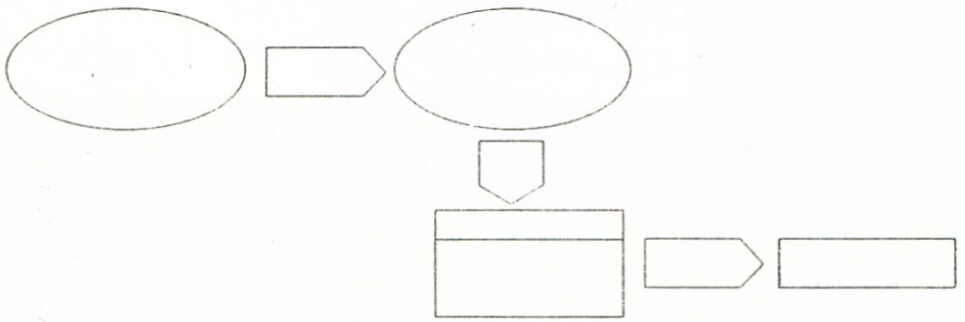
Nem szokás jelölni az AFD-ken a környezeti elemek közötti közvetlen adatfolyamokat, amelyek tehát nem lépnek be a vizsgált rendszerbe. A jobb érthetőség érdekében kivételesen fordul elő ilyen jelölés. Az ilyen adatfolyamokat a világos megkülönböztetés érdekében szaggatott nyíllal kell ábrázolni.

Egy környezeti elem lehet egyidejűleg forrás is és címzett is.

Az adattárakhoz hasonlóan a környezeti elemet is fel lehet tüntetni ugyanazon diagram több pontján is, ha ezzel elkerüljük a vonalak kuszaságát. Itt is jelölni kell a többszörözést, mégpedig a szimbólumba helyezett ferde vonallal, ahogyan az a 24. ábrán látható (ügyfél).

Anyagfolyam:

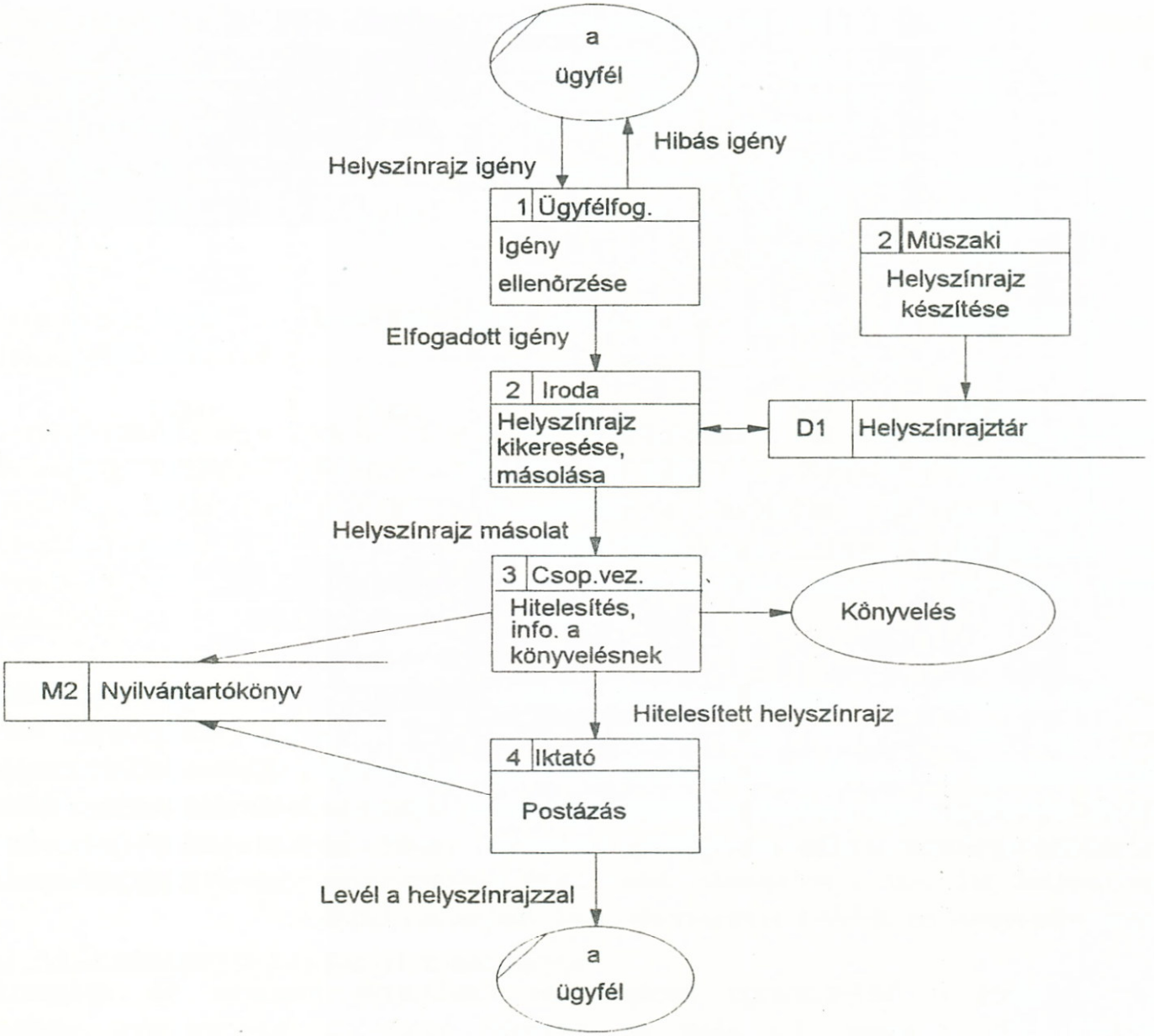
Anyagok fizikai áramlása csak ritkán jelölendő, hiszen diagramjaink alapvetően adat-, nem pedig anyagfolyam diagramok. Olykor mégis segítheti a megértést, ha egy-egy ponton jelöljük az adatokkal párhuzamosan áramló anyagokat. Erre a célra szolgál a 19./e ábrán látható jelkép, melynek használatát a 22. ábra mutatja be.



23. ábra

Példa a jelképek alkalmazására

A 24. ábra az ún. helyszínrajz–szolgáltatás egyszerűsített folyamatán keresztül mutatja be a jelképek használatát. A diagram összefoglalóan ábrázolja a folyamatot, anélkül, hogy részleteket tartalmazna. Az ilyen átfogó jellegű diagramokat magas szintűnek, ill. áttekintőnek is szokás nevezni



24. ábra

Az AFD-k elemei - mint láttuk - kapcsolatban állnak egymással, mégpedig magukon az adatfolyamokon keresztül, amit a diagramon az összekötő nyilak ábrázolnak. A diagramokon azonban nem létesíthetünk kapcsolatot bármely két elem között a nyilak alkalmazásával. A 25. ábra táblázatba foglaltan mutatja be a lehetséges, ill. a tiltott összekapcsolásokat.

A megszorítások értelme az, hogy kifejezésre juttassa: a tárolókezelés (akár adat-, akár anyag-) csak - a rendszer határain belüli - művelet segítségével történhet.

Megengedett AFD kapcsolatok

	Környezeti elem	Folyamat	Adattár	Anyag tároló
Környezeti elem	Csak külső adat- ill. anyagáram	IGEN	NEM	NEM
Folyamat	IGEN	IGEN	IGEN	CSAK anyagfolyam
Adattár	NEM	IGEN	NEM	NEM
Anyag tároló	NEM	CSAK anyagfolyam	NEM	NEM

25. ábra

9.3 Az adatfolyam diagramok szintjei

DeMarco írja az adatfolyam modellezést megalapozó könyvében: *"Amikor egy rendszer túl nagy ahhoz, hogy egyetlen lapon ábrázolhassuk, akkor folyamatait részrészfolyamatokra, valamint ezek közötti adatfolyamokra bonthatjuk fel. Ezt a felbontást akár több hierarchiaszinten át folytathatjuk. végeredményül az AFD-k szintekrebontott halmazát kapjuk."*[6]

A 24. ábra a helyszínrajz szolgáltatás rendszerét mutatta be, mégpedig összefoglaló jelleggel. Ez alatt az értendő, hogy az információs rendszer valamennyi fő összetevője szerepel rajta:

- a környezet,
- a folyamatok,
- az adattárak és
- az adatfolyamok.

Ez jelenti a felülről lefelé haladó rendszerleírás felső csúcsát, ahol tehát az elemzés elkezdődik. Tekintettel arra, hogy a diagramok méretét a könnyű áttekinthetőség és tárolhatóság érdekében A4 formátumra korlátozzuk, és egy lapon legfeljebb 7 folyamatot ábrázolunk, ebből az következik, hogy a

[6]DeMarco: Structured Analysis and System Specification, New York, Yourdon Press, 1978

"folyamatok" ilyenkor csak az alkalmazott jelkép miatt nevezhetők annak. Valójában — a teljes rendszer méretének függvényében — sokkal inkább nevezhetők alrendszereknek, fő funkcióknak vagy eljáráscsoportoknak.

Az adattárak közül csak azok tüntetendők fel ezen a szinten, amelyeket az itt ábrázolt funkciók közül legalább kettő használ (tehát a közösen használt adattárakat).

Természetszerűleg a funkciók belső adatáramlása sem jelenik meg adatfolyam alakjában.

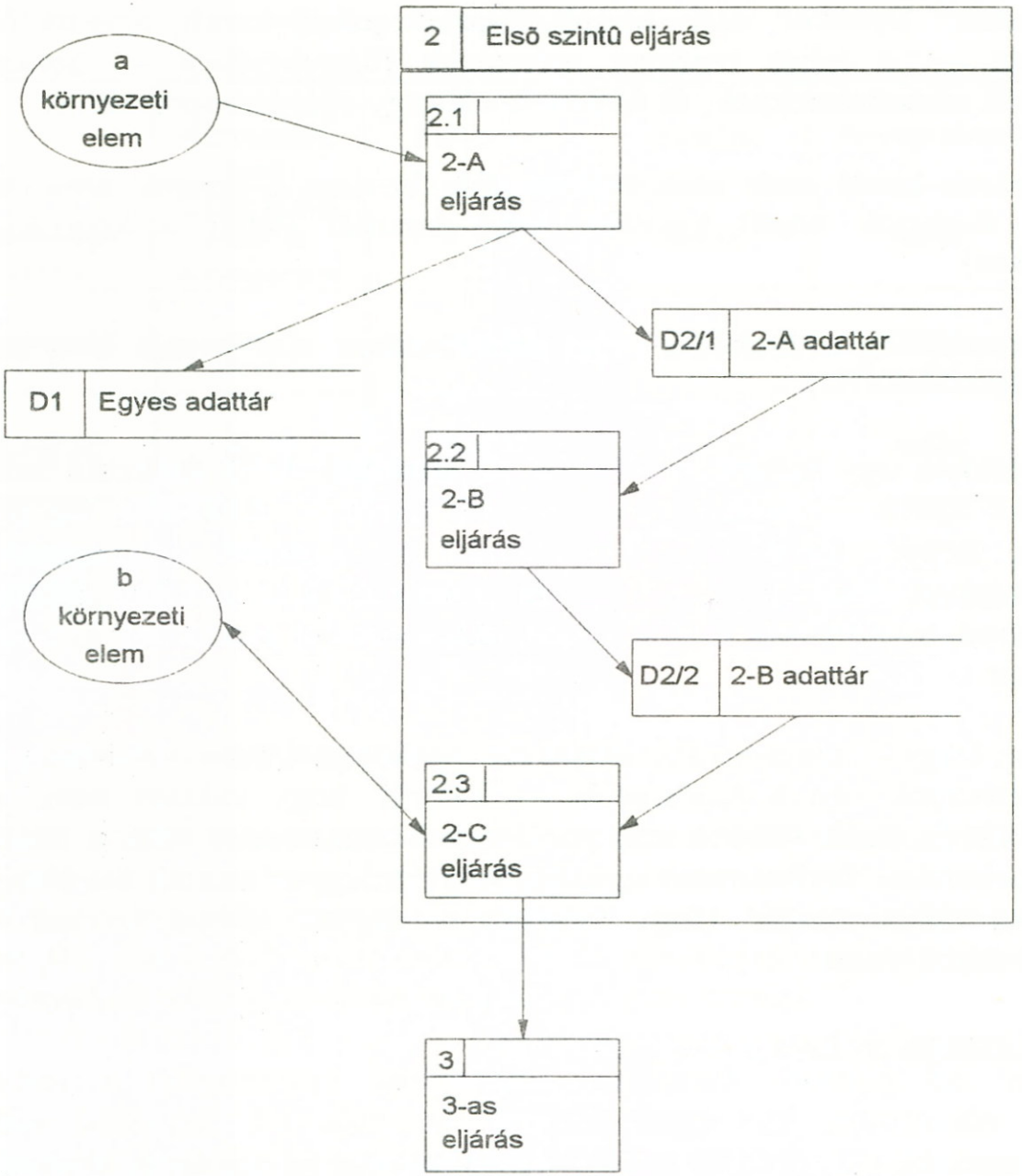
A részletekhez úgy tudunk eljutni, ha az áttekintő AFD funkcióit külön–külön ábrázoljuk újabb — mostmár alacsonyabb szintű — AFD-ken. Ha ilyenkor az derül ki, hogy vannak olyan "funkciók" az áttekintő diagramon, amelyek tulajdonképpen elemi eljárások (ezek meghatározását ld. később), amelyeket már nem bontunk tovább, míg másokat igen, akkor rosszul alakítottuk ki a felső szintű funkciókat.

Előfordul, hogy az alacsonyabb szintű AFD-k készítésekor — mivel ilyenkor már több ismeretünk van a rendszerről — kiderül, hogy valamit nem, vagy nem helyesen ábrázoltunk felsőbb szinten. Ilyenkor szükségessé válik a felsőbb szintű AFD módosítása. Nyilvánvaló ugyanis, hogy az egyes szintek diagramjai között nem lehet ellentmondás. Hogy valóban ne legyen, ahhoz be kell tartanunk bizonyos szabályokat.

Az AFD szintjeivel kapcsolatos szabályok

1. A magasabb szintű AFD minden egyes eljárása külön–külön fejtendő ki egy–egy alacsonyabb szintű diagramon. Ez azonban nem jelenti azt, hogy ha egy diagram valamelyik folyamatát tovább részletezzük egy alacsonyabb szintű diagramon, akkor ezt a magasabb szintű diagram valamennyi folyamatára nézve kötelező lenne megtennünk.

2. Az alacsonyabb szintű AFD-n szereplő eljárásokat a magasabb szintű eljárás azonosítójának decimális rendszerű alátörésével azonosítjuk. Ha pl. az eredeti, áttekintő szintű eljárások közül a 2. számút részleteztük, akkor az itt szereplő eljárások azonosítói rendre 2-vel kezdődnek: 2.1, 2.2, stb. Ezt a számozási konvenciót figyelhetjük meg az eljárások lebontásának lényegét bemutató 26. ábrán.



26. ábra

3. A részletező szintű AFD-k készítéséhez olyan nyomtatványt használunk, amelyen egy keret látható. Ebbe rajzoljuk a diagramot, a kereten kívülre pedig vagy felsőbb szintű elemek (pl. adattár), vagy környezeti elem ábrázolható a kapcsolódások könnyebb áttekinthetősége érdekében. A 26. ábrán a magyarázat kedvéért úgy készítetem el a második szintű diagramot, hogy az belekerült a kinagyított első szintbeli folyamat-szimbólum belsejébe. Ez egyúttal meg is adta azt a keretet, amelyen kívül a kapcsolódásokat feltüntetjük.

4. Ha vannak olyan adattárak amelyek csak a részletező diagramon jelennek meg, akkor ezeket a kereten belül ábrázoljuk, azonosításukkor pedig betűjelüket követően először annak a magasabb szintű eljárásnak a számát írjuk, amelyet éppen részletezünk az alacsonyabb szinten, majd "/" jel és egy, a szinten belüli sorszám következik, ahogyan ez a 26. ábrán is látható.

5. Az olyan adattárakat, amelyek már a magasabb szinten is megjelentek, de ezen a szinten is ábrázolni akarunk, a kereten kívülre kerülnek. Ugyanide rajzolandók az azonos szinten, de több diagramon is használt adattárak jelképei.

6. Ha környezeti elemmel való kapcsolatot kívánunk ábrázolni, az csak a kereten kívülre rajzolható, még akkor is, ha az adott elem kizárólag a részletező AFD eljárásával áll kapcsolatban.

7. A részletező diagram belépő, ill. kilépő adatfolyamainak meg kell egyezniük a magasabb szintű diagram megfelelő eljárásának be-, ill. kilépő adatfolyamaival. Ezt egyensúly-szabálynak is szokás nevezni.

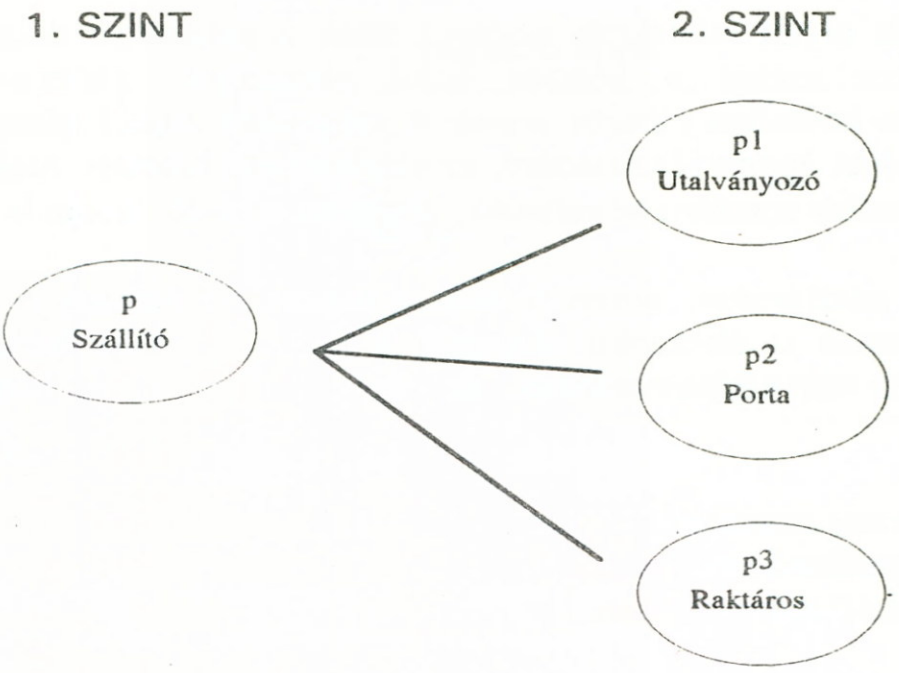
8. A részletező diagramokon egyes elemek szükség szerint felbonthatók a magasabb szintű diagramhoz képest. Ilyenkor az azonosító megfelelő alátörésével biztosítható a diagramok összhangja. A 27. ábra egy környezeti elem esetleges felbontásakor alkalmazható azonosítási szabályt mutatja be.

   Ezáltal megoldódik az a gond is, amikor az adatfolyam nyilát kellene elágaztatni, ami SSADM-ben nem megengedett. Ilyenkor vagy a forrás, vagy a címzett elem felbontását kell elvégezni és az így kapott új elemeket külön-külön adatfolyammal kell kapcsolni az eredetihez. Ezeknek megkülönböztetett nevet is kell adni. Ennek az esetnek tipikus példája, amikor egy többpéldányos bizonylat egyes példányai más-más címzetthez kell, hogy kerüljenek.

Ezekből a szabályokból látható, hogy olyan elemzési eszközhöz jutottunk, amely

- az egészből rész felé haladva dokumentálja a rendszereket (az egyre mélyebb részletek közötti verifikális kapcsolatot a hierarchikus felépítésű azonosítók biztosítják);

- minden felbontási szinten követni tudja az adott szint diagramjai közötti horizontális kapcsolatokat is (amit a részletező diagramok keret részén kívülre helyezett kapcsolódó elemekkel biztosíthatunk).



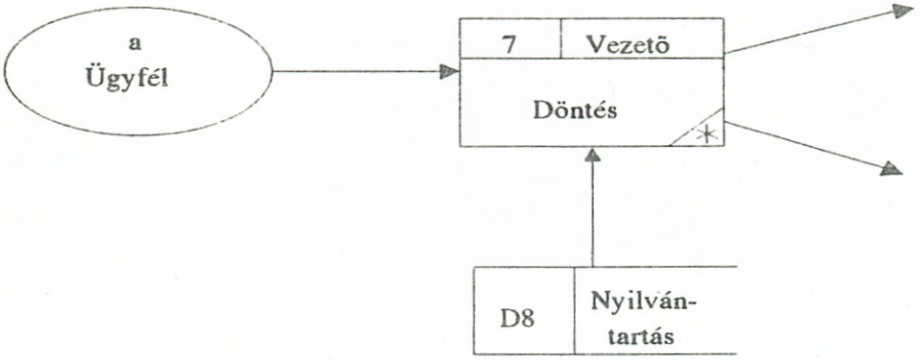
27. ábra

Ezek a tulajdonságok pontosan megfelelnek a klasszikus rendszerszemlélet követelményeinek, amelyek a gyakorlatban a strukturált módszertanokban, s ezen belül is legfőképpen az SSADM-ben öltöttek testet.

Elemi szintű folyamatok

Mielőtt arra a kérdésre válaszolnánk — amely az olvasóban már valószínűleg megfogalmazódott —, hogy vajon hogyan határozható meg a szükséges hierarchia-szintek száma, előbb tisztáznunk kell az ún. elemi szintű folyamatok fogalmát.

Nyilvánvaló, hogy a folyamatokat nem részletezhetjük a végtelenségig újabb és újabb hierarchia-szinteken. Előbb-utóbb elérünk ugyanis egy olyan szintet, ahol tartalmuk már olyan kis méretűre zsugorodik, hogy a hozzájuk kapcsolódó adatfolyamok teljesen mesterkéltté válnak, mert lényegében nincs megfelelőjük a valóságban. Már ezt megelőzően is jelentkezhet az a gond, hogy egy folyamatból kiinduló két vagy több adatfolyam tényleges megvalósulása valamilyen feltételtől függ. (ld. 28. ábra).



28. ábra

Nem mindig oldható meg, hogy a kilépő adatfolyamok neveinek ügyes megválasztásával egyértelművé tegyük a helyzetet.

Az adatfolyam diagramok döntési jelképet nem tartalmaznak, a döntést folyamatnak, vagy folyamat részének tekintjük. Következésképpen valamilyen, az AFD-től különböző leírásban kell megadnunk, hogy mi történik a tovább már nem bontott folyamatblokk belsejében. Az ilyen folyamatot *elemi szintű*nek nevezzük és a jelkép jobb alsó sarkába helyezett csillaggal jelöljük.

Az SSADM kerüli a hosszadalmas, bonyolult leírásokat, ezért azt a követelményt támasztja, hogy az elemi folyamat leírása ne legyen hosszabb egy A4-es oldal felénél.

A szükséges szintek száma

Azzal kell kezdenünk, hogy erre vonatkozó szabály tulajdonképpen nincs. Van azonban tapasztalat és vannak bizonyos tényezők, amelyek a szintek számát többé–kevésbé automatikusan behatárolják.

A tapasztalat azt mutatja, hogy háromnál több szintű leírást igénylő rendszer meglehetősen ritkán fordul elő, többnyire két szint már elegendőnek mutatkozik.

A szintek számát automatikusan behatároló tényezők a következők:

1. Egy A4 méretű diagramon nem tanácsos 7–10 folyamatnál többet ábrázolni, mert az ennél több folyamat — a kapcsolódó többi elemmel együtt — nehezen áttekinthetővé teszi a diagramot, ami viszont ellent mondana az SSADM alapelveinek.

2. A legalsó szintű diagramokat ekkor értük el, ha az elemi folyamatok leírása nem igényel több helyet, mint azt a fentebb kifejtettük.

Az egy diagramon ábrázolandó részletek mennyiségét még néhány további összefüggés is befolyásolhatja, amelyet érdemes szem előtt tartani:

- ügyeljünk arra, hogy túl sok részlet megadásával ne tereljük el a figyelmet a fő folyamatokról;

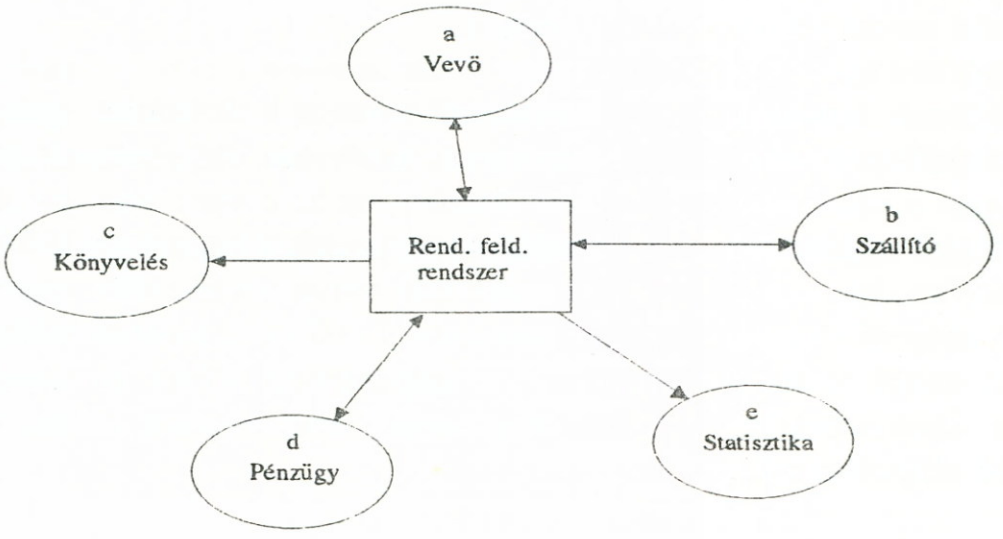
- úgy bontsuk szét a folyamatokat, hogy az így kapott részek között jól értelmezhető kapcsolat (adatfolyam) legyen;

- számítsunk arra, hogy a mélyebb szintek felé haladva a folyamatok száma egy-egy diagramon a felette levő szinthez képest csökken;

- azok a diagramok a jók, amelyek könnyen áttekinthetők — és ez a legfontosabb.

Olykor csábító lehet a gondolat, hogy egyetlen, nagyméretű AFD-n ábrázoljunk mindent. Ez azonban minden bizonnyal lerontatná az áttekinthetőséget a rajta szereplő elemek és összekötő adatfolyamok nagy száma miatt. Emellett arra is gondolni kell, hogy a dokumentáció minél könnyebben kezelhető legyen. Ezért cél az SSADM-ben az A4-es méret egységes alkalmazása.

Végül az AFD-k szintjeivel kapcsolatos, hogy az 1. szintű, vagy áttekintő diagram felett létezhet még egy szint (nem kötelező a készítése), amelyet *kapcsolat diagram*nak - idegen kifejezéssel kontextus diagramnak - nevezünk. Ezen a szinten az egész rendszer egyetlen folyamat-blokkba sűrűsödik, amelyet a környezeti elemek (tehát a rendszer környezete) vesznek körül (29. ábra)



29. ábra

9.4 Fizikai és logikai AFD

Az adatolyam diagramok elvének, jelölési szabályainak bevezetésekor, továbbá a diagramok szintekre bontásának magyarázatakor lényegében meglevő információs rendszert AFD-vel való leírásának módjáról volt szó. Ez korántsem jelenti azt, hogy tervezett rendszereket ne lehetne leírni AFD-vel.

A meglevő és a tervezett rendszerek között azonban szükség van valamilyen átmenetre. Gondoljuk csak át, miért is foglalkozunk meglevő információs rendszerek felmérésével, elemzésével és dokumentálásával. Természetesen semmilyen más cél nem vezet bennünket, mint a működési mód, a problémák és a következmények megismerése, hogy azután egy ennél remélhetőleg jobb rendszert tervezhessünk.

Amikor a működési mód megismeréséről beszélünk, akkor érdeklődésünk kettős:

1. Annak megismerése, ahogyan a meglevő rendszer a maga fizikai valóságában, tehát az adott eszközeivel működik. Erre azért van szükség, mert a megismerésnek szinte ez az egyetlen lehetséges kiindulópontja, továbbá a meglevő rendszer fizikai szintű feltérképezése tömegével hozhat a felszínre olyan problémákat, amelyek éppen az alkalmazott eszközök és a megoldandó feladatok közötti ellentmondásból fakadnak.

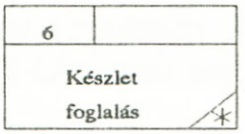
2. Az információs rendszer belső logikájának feltárása, amely független az alkalmazott eszközöktől. Ez is csak úgy lehetséges, ha előbb fizikai szinten ismerjük meg a rendszert. A meglevő rendszer logikai leírását a fizikai rendszer megfelelő átalakításával érjük el, aminek a folyamatát a következő pontban tekintjük át.

Amennyiben egy információs rendszer logikai leírását akarjuk elkészíteni, ehhez ismét felhasználhatjuk az AFD-k technikáját, de a diagramokat ilyenkor az eddigiektől kissé eltérő módon kell értelmeznünk, hogy megszabadítsuk a leírást a fizikai kötöttségektől. Ez a diagramok egyes elemeire vonatkozóan a következőket jelenti:

Logikai folyamat

Nevében nem tartalmazhat konkrét (fizikai) feldolgozási módra utaló kifejezéseket, tehát pl. nem alkalmazható a "Lefoglalt készlet felvezetése a nyilvántartó kartonon" megnevezés. Helyette általánosabb — de továbbra is kifejező — nevet kell használnunk, ami esetünkben lehet pl. "Készletfoglalás".

Ábrázolásbeli különbséget jelent, hogy a logikai diagramokon nem tüntetjük fel az folyamatért felelős szervezeti egység, vagy személy nevét, a megfelelő rovat tehát üresen marad (ld. 30. ábra).



30. ábra

Logikai adatfolyam

Ábrázolása nem változik a fizikaihoz képest, de nevében nem tartalmazhat az adatáramlás tényleges megvalósulási módjára való utalást. Ha pl. a fizikai diagramon egy adatfolyam neve "számla másodpéldánya", akkor a megfelelő logikai név "számlaadatok" lehet.

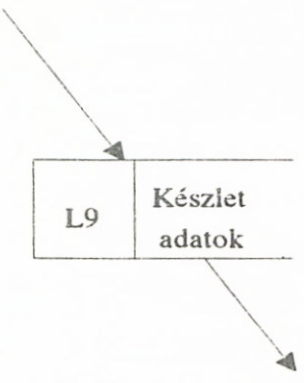
Logikai adattár

Ábrázolása annyiban változik, hogy az azonosítóban vagy nem szerepeltethetünk "M" betűjelzést (ami a manuális kifejezésre utalna), csak "D"-t, vagy pedig "L" betűvel jelezzük, hogy ún. logikai adattárról van szó. A "D", vagy "L" betű mellett természetesen ilyenkor is szerepel azonosító szám.

Az adattár neve itt sem tartalmazhat fizikai utalást.

A későbbiekben látni fogjuk, hogy a logikai adattár a rendszer adatmodelljének érintett részét, tehát egy adat-almodellt jelent.

Logikai adattárra mutat példát a 31. ábra.



31. ábra

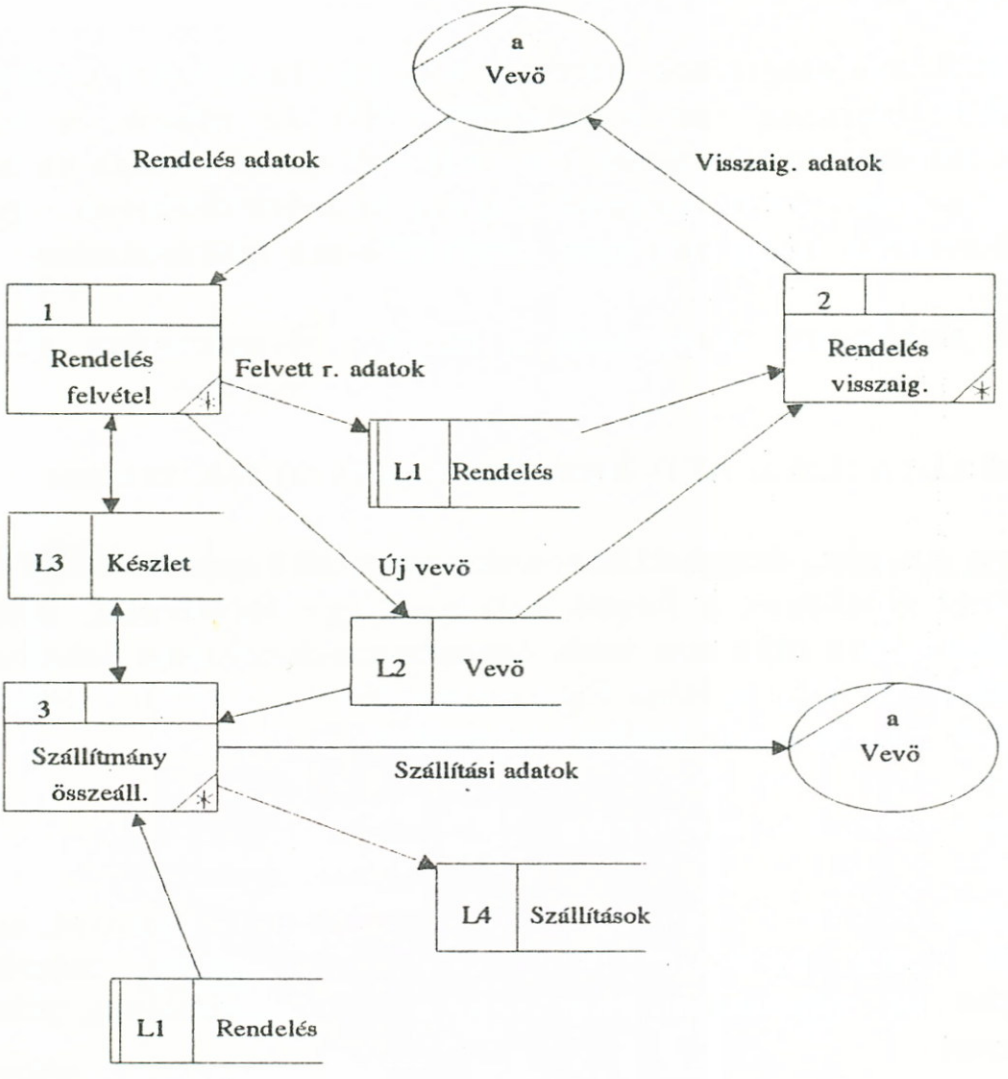
Környezeti elem

Változatlan alakban fordul elő a logikai AFD-ken, aminek az az oka, hogy a környezetet adottnak tekintjük, tehát az ábrázolásban se legyen megkülönböztetés.

Anyagfolyam

Logikai AFD-n anyagfolyam nem szerepelhet.

Logikai adatfolyam diagramra mutat példát a 32. ábra



32. ábra

Logikai AFD kétféleképpen készülhet:

- fizikai diagram(ok)ból, vagy

- fizikai előzmény nélkül.

Az utóbbi eset nem gyakori, olyankor fordul elő, ha programcsomag tervezését végezzük, vagy újonnan létesült szervezet egyedi fejlesztésű információs rendszerét kell elkészíteni. Ezekben az esetekben közvetlenül alkalmazhatók a logikai diagramokkal kapcsolatban fentebb ismertetett szabályok.

Bonyolultabb a helyzet akkor, ha egy, már elkészült fizikai AFD-halmazból kell levezetnünk az ennek megfelelő logikai diagramcsoportot. Ilyenkor az a célunk, hogy kiszűrjük a meglevő rendszernek az alkalmazott eszközök, valamint a változó vállalati politikák, ill. egyszerűen az idő múlása révén elfedett lényegét és ezt fogalmazzuk meg a logikai diagramokban.

Amíg tehát a fizikai diagramok készítése során a "HOGYAN történik?", most a "MI történik?" kérdésén van a hangsúly. Mik az adatok és mik azok az átalakítások, amiken az adatoknak keresztül kell menni? Maga az az átalakítási folyamat is - melynek során fizikaiból logikai AFD-ket készítünk - gyakran vezet el a meglevő rendszer addig még nem ismert hibáinak felfedezéséhez.

A logikaivá alakításnak megvannak a maga - receptszerűen alkalmazható - lépései:

1. Az átalakítást a fizikai AFD-hierarchia legalsó szintjén kezdjük.

   Ennek az az oka, hogy ilyen módon egy alsó szintű diagram logikaivá alakításával előállítjuk a felette levő szint egy folyamatát, tehát a legalsó szintek átalakítása után már többé-kevésbé mechanikusan haladhatunk felfelé a hierarchiában. Úgy is lehet fogalmazni, hogy a felsőbb szinteket már nem *átalakítjuk*, hanem a már logikai alsó szintből *állítjuk elő*.

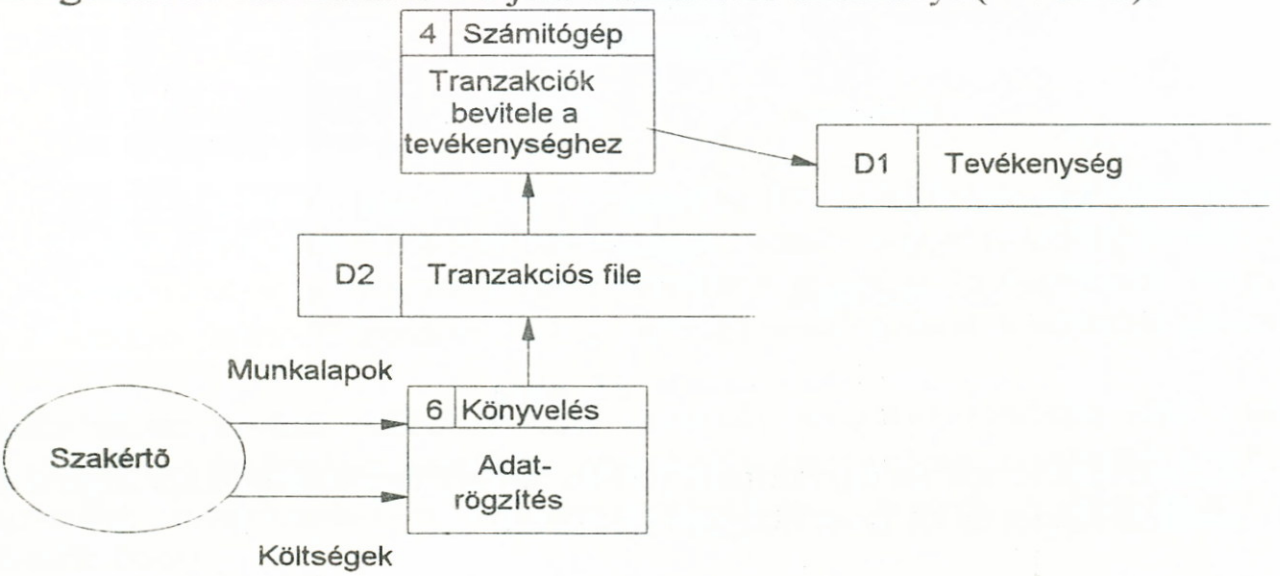
2. Alakítsuk logikaivá az adattárakat.

   Ez egyrészt azt jelenti, hogy távolítsuk el azokat az adattárakat, amelyek csak az adott fizikai környezet miatt váltak szükségessé, másrészt vegyük figyelembe az ugyanezen projekt keretében végrehajtott adatmodellezés eredményeit.

   A fizikai környezet korlátai gyakran teszik szükségessé olyan adattárak létesítését, amelyeknek egyébként semmi közük sincs az üzleti folyamatokhoz. Ezek általában az adatfeldolgozási műveletek végzésének adott keretei között elkerülhetetlen átmeneti tárolást végeznek.

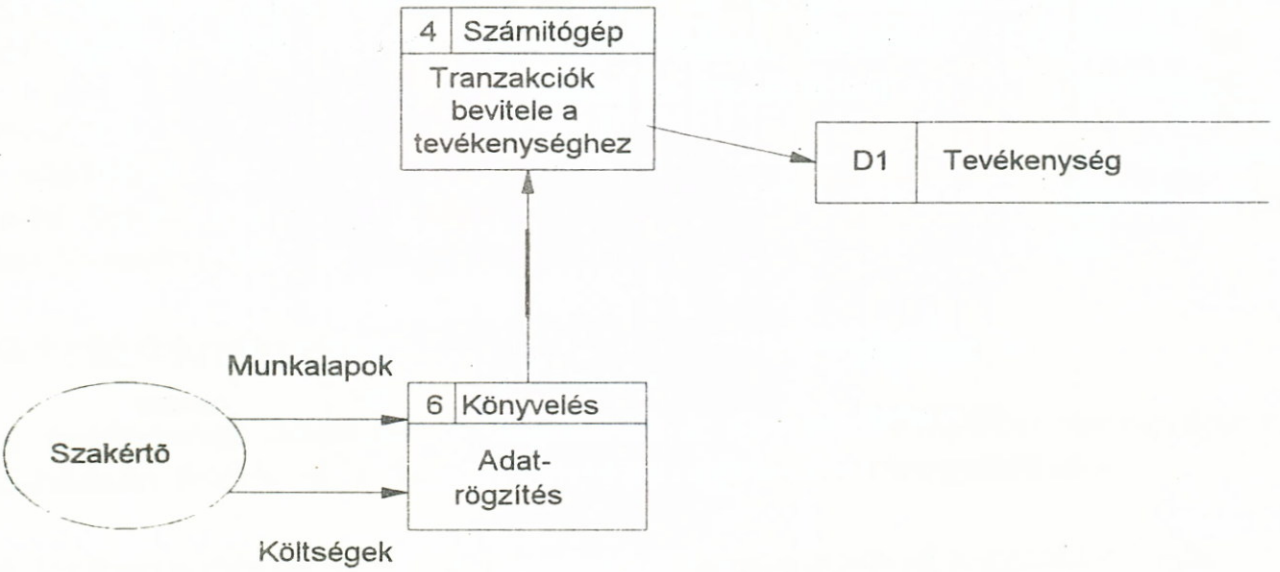
Ilyen esetre látunk példát a 33. ábrán. Olyan környezetről van szó, ahol egy tanácsadó cég szakértői projekteken dolgoznak - bérmunkában. Munkaidejüket és felhasznált költségeiket adatlapokon közlik, amelyeket a jelenlegi rendszerben központi számítógéppel, kötegelt módban visznek be az érintett projekt-tevékenységekhez.

A logikaivá alakítás első lépéseként eltávolítottuk a jelenlegi kötegelt feldolgozásban használatos D2 jelű tranzakciós állományt (34. ábra).



33. ábra

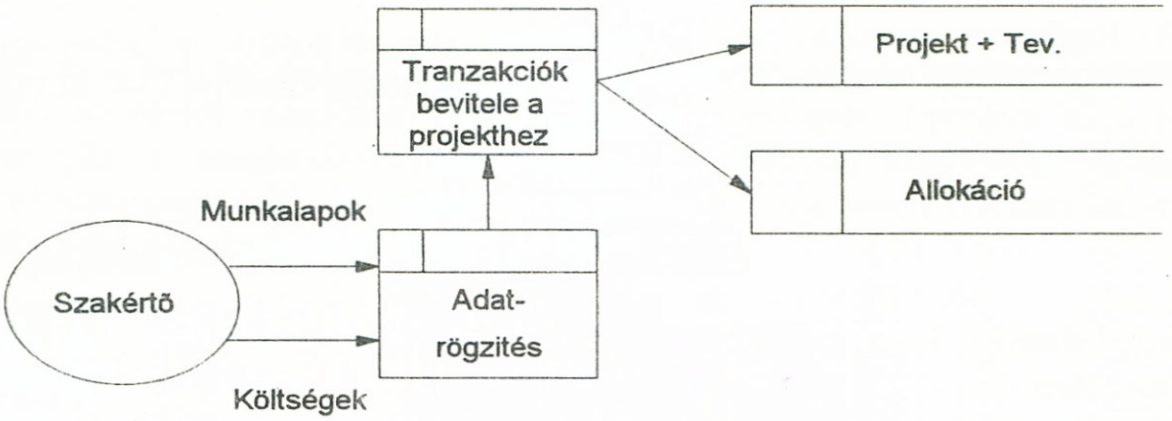
Nyilvánvaló, hogy logikai szempontból a D2 adattárra nincsen szükség.



34. ábra

A logikai adatmodellezés eredményeinek figyelembevétele azt jelenti, hogy a megmaradt fizikai adattárakat az adatmodellnek az ábrázolt folyamatokkal

kapcsolatos részével (almodelljével) helyettesítjük. A jelölés a diagramon továbbra is a megszokott adattár-jelképpel történik, csak ennek belső tartalmaként feltételezzük az említett almodellt. Sok esetben megtehetjük, hogy olyan adattárakat tüntetünk fel a logikai AFD-n, amelyek egy-egy adatmodellbeli egyedtípusnak felelnek meg (ld. 35. ábra)

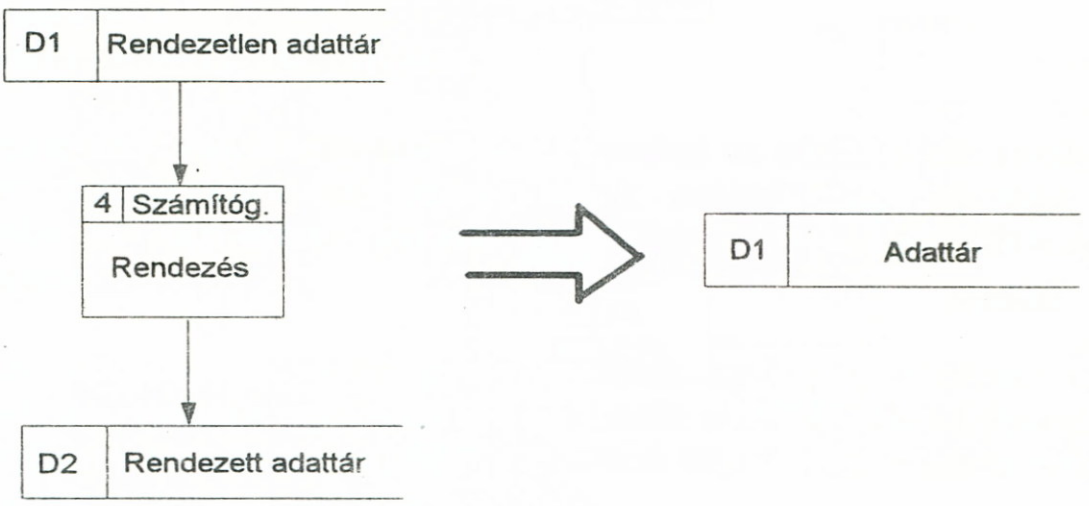


35. ábra

Az Allokáció azt az egyedtípust jelenti, amely egymáshoz kapcsolja a projekt tevékenységet és az azon dolgozó szakértőt.

3. Alakítsuk logikaivá a folyamatokat.

A folyamatok átalakítása során eltávolítjuk azokat, amelyek az adatok újraszervezését végzik, mert az újraszervezés szükségessége az alkalmazott eszközök korlátaiból származik. Logikai szempontból azt feltételezzük, hogy az adatmodell szerkezete minden szükséges elérési igényt kielégít (ld. 10. fejezetben). Erre az átalakításra ad példát a 36. ábra.



36. ábra

Az azonos feladatot megoldó folyamatokat összevonjuk (37. ábra).

4. Alakítsuk logikaivá az adatfolyamokat.

5. Vizsgáljuk felül és szükség szerint módosítsuk a kapott eredményt.

6. Készítsük el a magasabb szintek logikai diagramjait.

9.5 Az AFD készítés gyakorlata

Egy információs rendszer adatfolyam diagramjai többféleképpen is elkészíthetők. Ebben a fejezetben olyan módszert ismertetünk, amely a gyakorlatban bevált. Az olvasó - némi gyakorlat megszerzését követően - maga is rá fog jönni, hogy a "Hogyan kezdjek hozzá?" kérdése csak a valóban kezdő számára jelenthet gondot.

A munka szinte kivétel nélkül mindig valamely meglevő rendszer első szintű diagramjának elkészítésével kezdődik. Ilyenkor a szervezőnek a vizsgált rendszer legalapvetőbb tulajdonságaira kell összpontosítania, melyek a következőkben foglalhatók össze:

- a rendszer határai,
- a környezeti elemek (adatforrások és információ-igénylők)
- a rendszer legfontosabb bemeneti és kimeneti adatfolyamai (amelyek tehát a rendszer és környezete között áramlanak),
- a rendszer fő funkciói (esetleg alrendszerei — a méret függvényében).

A rendszer ilyen jellegű behatárolása projekttervezési szempontból is lényeges, mert segítséget nyújt a rendszer méretének megállapításában. Ez, valamint a funkciók felosztása alapot képezhet azokhoz a becslésekhez, melyek célja a munkaerő-szükséglet megállapítása és a munkacsoport(ok) felállítása.

Az áttekintő diagram elkészítésének lépései a következők:

1. A rendszerben használt dokumentumok (nyomtatványok) összegyűjtése, a kitöltésért felelős ill. a fogadó szervezeti egység(ek) megjelöléséve.

2. A legfontosabbnak ítélt dokumentumok kiválasztása és jegyzékbe foglalása.

3. Ún. dokumentumáramlási diagram elkészítése, amelyen a csomópontok szervezeti egységeket jelképeznek, az összekötő nyilak pedig az áramló dokumentumokat.

4. A tovább elemzendő rendszer határainak kijelölése (berajzolása) a dokumentumáramlási diagramon, a felhasználóval közösen.

5. Az így kijelölt határon belül helyet foglaló szervezeti egységek eljárásokkal való helyettesítése a szükségessé váló további adatáramlások feltüntetésével.

6. Az AFD ellenőrzése egyértelműség, teljesség és az eljárások helyes kijelölése szempontjából.

7. Az AFD felülvizsgálata a felhasználóval közösen és az esetleg szükségessé váló javítások átvezetése.

Nem szabad elfeledkezni arról, hogy bár a felhasználóval való együttműködést csak a 4. és 7. lépénél említettük meg, ez tulajdonképpen folyamatos a felmérések és az interjúk során.

Az 1. szintű AFD átfogó képet ad a vizsgált rendszerről, de nem mutatja a részleteket. Ez tökéletesen megfelel annak, ahogyan a felmérés során a rendszerszervező dolgozik és a munka időbeli előrehaladásával is összhangban van. Innen - ahogy a részletek egyre ismertebbé válnak - elkészíthetők a részletesebb diagramok, ahogyan a 9.3 pontban láttuk.

Ekkor tartunk ott, hogy áttérhetünk a fizikai AFD-kről a logikaiakra. Az áttérés lépéseit a 9.4 pont tartalmazza.

10. fejezet
Logikai adatmodellezés

10.1 A logikai adatmodellezés helye és szerepe

A modern strukturált módszertanokban a folyamatok modellezése (elemzése és tervezése) mellett a másik alapvető terület az adatok modellezése. Szükségessége akkor vált nyilvánvalóvá, amikor a rendszerszervezés fejlődése során kiderült, hogy az adattárolást már nem lehet programközpontosan végezni, mint korábban, mert ugyanazokat az adatokat egyre több programnak kell használnia. Ekkor a gondolkodás egy magasabb absztrakciós szinten folytatódott, az adatok feldolgozásoktól független, belső logikáját keresve, és ez vezetett el az adatmodellezés szakterületének kifejlődéséhez.

Sokféle modellezési koncepció jelent meg, ezek közül azonban hamarosan kiemelkedett két irányzat:

- egyed-tulajdonság-kapcsolat[7] (ETK) modellezés (Bachman)
- relációs modellezés (Codd)

A két irányzat először mint két, egymással versengő adatbáziskezelő rendszer-koncepció jelent meg. Nemzetközi szakmai fórumokon igen érdekes vita folyt Bachman és Codd között, míg végül kiderült, hogy a két közelítésmód tulajdonképpen azonos. Ez a felismerés időben körülbelül egybeesett a strukturált módszertanok körvonalazódásával, így épült be az adatmodellezés az új módszertanok technikái közé.

Az SSADM ezen a téren abban "alkotott nagyot", hogy a fenti két közelítésmódot olyan két technikaként értelmezte, amelyek egymástól függetlenül alkalmazhatók, végeredményeik pedig összehasonlítható alakban megjelenve alkalmasak egymás ellenőrzésére. Ezáltal az információs rendszerek logikai szintű adattervezését rendkívül szilárd alapokra sikerült helyezni.

[7] A fogalmak magyarázatát ld. később ebben a fejezetben

Az ETK-modellezés eredeti koncepciója szerint első lépésben egyed-kapcsolati (EK) modellet kell megalkotni és később - amint részletesebb ismeretekre teszünk szert a rendszerről - kell kibővíteni. A modellezésnek ez a lépése az, ami az SSADM ú.n. Logikai AdatModellezési (LAM) technikáját alkotja.

A logikai adatmodellezés célja:

Valamely szervezet (vállalat, intézmény) működéséhez szükséges adatok feldolgozási követelményektől független és feldolgozó, ill. tároló eszközök lehetőségei által nem korlátozott leírása, belső, logikai szerkezetének meghatározása.

Ezt a modellt csak egy dolog korlátozza - a valóság, amelyet a maga sajátos eszközeivel vissza kell tükröznie.

Ehhez a legfontosabb célkitűzéshez kapcsolódik még néhány további cél:

- Segítsen az alkalmazási terület minél jobb megértésében a fejlesztés valamennyi résztvevője (felhasználók, szervezők, programozók és egyéb szakemberek) számára.

- Szolgáljon alapul az adattárolás konkrét megoldásának megtervezéséhez.

10.2 Az adatmodell elemei

Egyed

Az alkalmazott modellezési elv abból indul ki, hogy egy szervezet megfelelő működtetéséhez a szervezetet alkotó embereknek tudniuk kell egy sereg *dologról*. Pl. tudniuk kell a szervezet által gyártott, vagy forgalmazott cikkekről, vagy a szervezet dolgozóiról, a szervezet által intézett ügyekről, stb. Ezeket a dolgokat kicsit tudományosabban objektumoknak, vagy másképpen egyedeknek nevezzük. *Egyednek tekinthetünk* tehát *bármit - konkrét, vagy akár elvont tárgyat, ill. fogalmat - amiről tudnunk kell valamit, vagy valamiket.* Úgy is fogalmazhatunk, hogy számunkra minden olyan tárgy egyed - és így adatmodellünk részévé válik - amelynek egy, vagy több tulajdonságát a szervezetben ismernie kell valakinek, vagy valakiknek.

Minden modellezés általánosításra törekszik, ami esetünkben azt jelenti, hogy az adatok logikai szerkezetét nem konkrét, fizikailag létező tárgyak konkrét tulajdonságaival hozzuk kapcsolatba, hanem a tárgyakat bizonyos kategóriákba soroljuk. Egy személyzeti tárgyú adatrendszer *modellezésekor* pl. nem Szabó

János konkrét adataira vagyunk kíváncsiak, hanem a SZEMÉLY, mint objektum leírásához szükséges tulajdonságok fajtáiban vagyunk érdekeltek (AZONOSÍTÓ, NÉV, CÍM, stb.).

Emiatt teszünk különbséget **típus** és **előfordulás** között. Van tehát *egyedtípus* és *egyed előfordulás*. A SZEMÉLY a típus és Szabó János ennek egy előfordulása. a logikai adatmodell diagramján az egyedtípusok ábrázolása a 37. ábra szerinti lekerekített sarkú téglalappal történik, melybe beírjuk az egyedtípus nevét.

37. ábra

Kapcsolat

A valóságos világ objektumai, tehát az egyedek közötti viszony. Kapcsolat állhat fenn két egyedtípus között, vagy ugyanazon egyedtípus két előfordulása között. Ez utóbbi esetben egy egyedtípus önmagával létesített kapcsolatáról beszélünk. A kapcsolatok esetében ugyanúgy megkülönböztetjük a kapcsolattípus és a kapcsolat előfordulás fogalmát, mint az egyedek esetében.

A logikai adatmodellek diagramjain egyedtípusokat és kapcsolattípusokat ábrázolunk, ez utóbbiakat az egyedtípusok már megismert szimbólumai közötti vonalak segítségével.

38. ábra

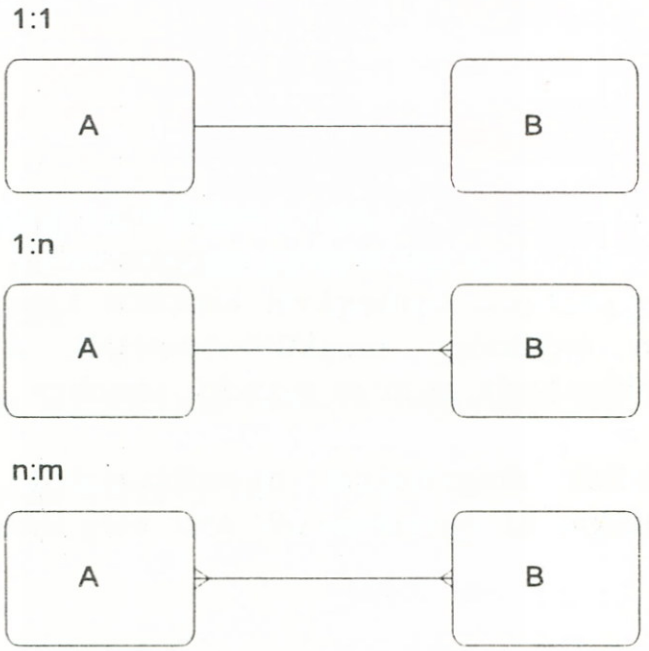
Van azonban a kapcsolattípusnak néhány fontos tulajdonsága, amelyek a modellezés későbbi menetét jelentősen befolyásolják, ezért a diagramon is fel kell ezeket tüntetni. Ilyen a kapcsolat *foka*, valamint a kapcsolat *jellege*.

A kapcsolat fokán azt értjük, hogy előfordulások szintjén milyenek a mennyiségi viszonyok a kapcsolódó egyedek között, vagyis, hogy az egyik egyedtípus egy előfordulásához mennyi kapcsolódhat a másikból. Persze, ha ez egynél több,

akkor csak annyit tudhatunk, hogy sok. Ezen az alapon tehát három eset lehetséges:

- egyhez egy,
- egyhez több és
- többhoz több.

A harmadik esetet úgy kell felfognunk, hogy ekkor az egyik egyedtípus egy előfordulásához a másik egyedtípus sok előfordulása kapcsolódhat, és ez fordítva is igaz. A három esetre úgy is lehet hivatkozni, mint 1:1, 1:n és n:m tipusú kapcsolatra. Ezeket ábrázolástechnikailag a 39. ábrán látható módon különböztetjük meg.



39. ábra

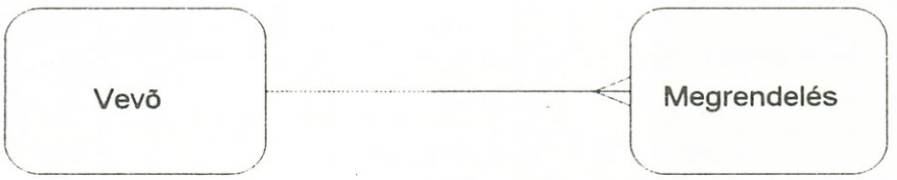
Az ábrából világosan látható, hogy a "több" oldalán elágaztatjuk a vonalat és így seprűszerűvé válik. Az 1:n kapcsolatban *fölérendelt* és *alárendelt* egyedtípust különböztetünk meg.

A kapcsolat jellege alatt azt értjük, hogy mi az összefüggés egy egyedelőfordulás léte és aközött, hogy az adott kapcsolat fennáll-e. Másképpen fogalmazva: létezhet-e az egyed anélkül, hogy létezne az adott kapcsolatban hozzá kapcsolt másik egyed, vagy nem.

Vegyünk egy egyszerű példát, ahol a VEVŐ-MEGRENDELÉS kapcsolatról van szó. A vevőinket akkor is célszerű nyilvántartanunk, ha éppen nincs élő rendelésük

nálunk, vagyis úgy is fogalmazhatunk, hogy a VEVŐ egyedtípus egy előfordulása akkor is létezhet, ha nem tartozik hozzá egy MEGRENDELÉS sem. Ha viszont azt a kérdést akarjuk eldönteni, hogy egy MEGRENDELÉS előfordulás létezhet-e anélkül, hogy az azt feladó vevőt nyilvántartanánk, akkor a válasz egyértelműen *nem*. Ha ugyanis ezt megengednénk, akkor előfordulhatnának az adatbázisunkban olyan rendelések adatai, amelyekről nem tudnánk kideríteni, hogy melyik vevőnkhöz tartoznak.

A kapcsolattípusok jellegét a kapcsolatot jelképező vonal folyamatos, vagy szaggatott voltával ábrázoljuk, amint az a 40. ábrán a fenti példára vonatkoztatva látható.

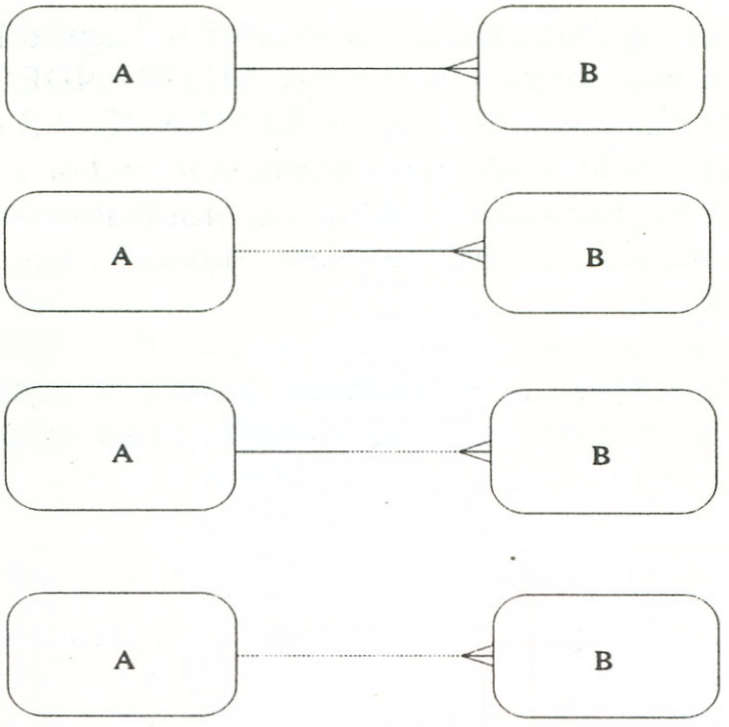


40. ábra

Annak az egyedtípusnak az oldalán, amelyik számára "nem létkérdés" a másik létezése, a kapcsolat vonala meg van szaggatva, míg a másik oldalon, ahonnan nézve a kapcsolatnak meg kell lennie, a vonal folyamatos.

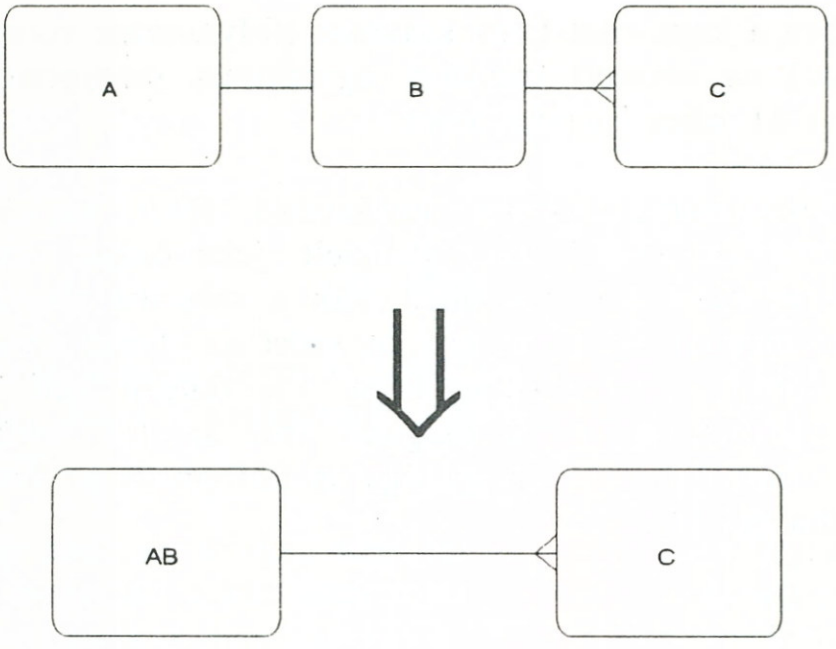
Ennek megfelelően a kapcsolat lehet kötelező (folyamatos vonal), vagy esetleges (szaggatott vonal) az érintett mindkét egyedtípus szempontjából. Az összes lehetséges esetet a 41. ábra foglalja össze.

A 39. ábrán bemutatott három különböző fokú kapcsolat modellezési szempontból nem azonos értékű. Kiemelt jelentősége van az 1:n fokú kapcsolatoknak, aminek a pontos okát csak a relációs elemzés technikájának ismeretében lehet megadni. Kérem, érje be most az olvasó azzal, hogy a logikai adatmodellezés során minden kapcsolatot 1:n fokúra kell átalakítani, vagy visszavezetni. Viszonylag könnyű dolgunk van az 1:1 fokú kapcsolatokkal, ezekről ugyanis könnyű belátni, hogy úgy is felfogható az ezekben érintett két egyedtípus, mintha egy lenne.



41. ábra

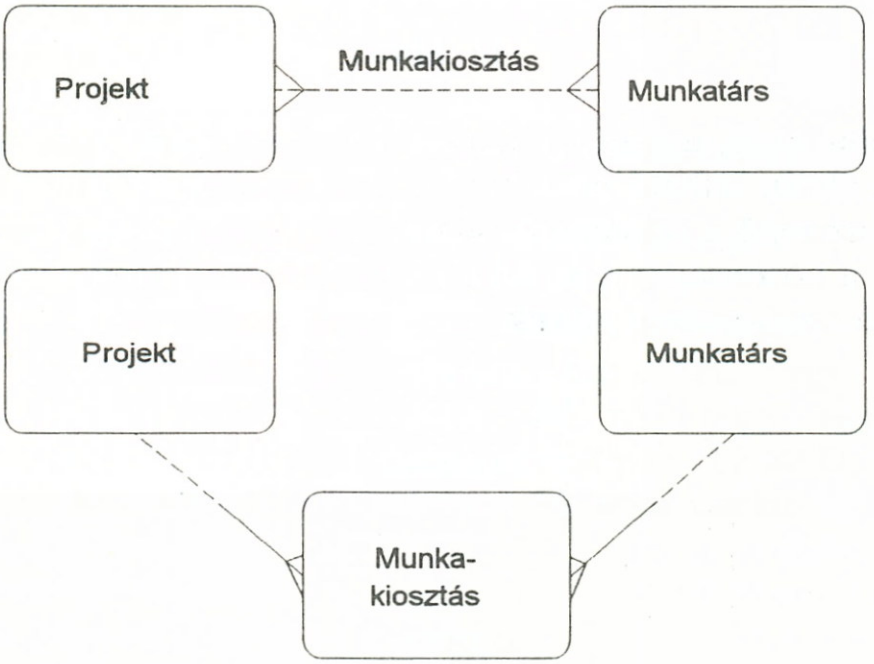
Az 1:1 fokú kapcsolat átalakítását a 42. ábra mutatja be.



42. ábra

Hasonlóképpen foglalkozni kell az n:m fokó kapcsolatok átalakításával. Ennek egy igen egyszerű, mechanikus módszere van: Eltávolítjuk a kapcsolatot,

beillesztünk a helyére egy új, ún. kapcsoló egyedtípust, amelyet mind a két eredeti egyedtípussal összekapcsolunk egy-egy 1:n fokú kapcsolattal, mégpedig úgy, hogy mind a két új kapcsolattípus "sok" oldala a kapcsoló egyedtípusnál legyen (43. ábra).



43. ábra

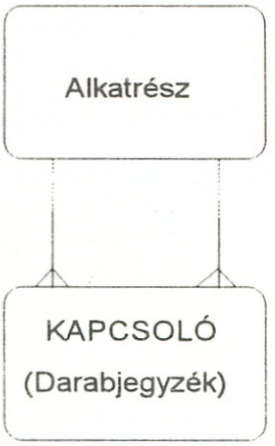
Az eredeti kapcsolat azt fejezte ki, hogy egy projekten sok munkatárs dolgozhat és fordítva. Az átalakítás mechanikus alkalmazása után könnyű felfedezni, hogy a létrehozott kapcsoló egyedtípusnak hasznos jelentéstartalmat tulajdoníthatunk: leírja a projektek és munkatársak egymáshoz rendelését.

Megjegyzendő, hogy a kapcsoló egyedtípus oldalán az új kapcsolatok kötelezőek, ami abból következik, hogy a kapcsoló létének feltétele, hogy legyen, amit összekapcsol. Míg tehát a fenti példában az eredeti kapcsolat mindkét irányban esetleges volt, addig a kapcsoló egyedtípus oldalán mindkét új kapcsolat kötelező.

A kapcsolatok egyik különleges esete az, amikor egy és ugyanazon egyedtípus különböző előfordulásai állnak kapcsolatban egymással. Az ilyen kapcsolatot *rekurzív kapcsolat*nak nevezzük és a 44. ábrán látható módon ábrázoljuk.

44. ábra

Az ilyen kapcsolatoknak ugyanúgy három féle foka lehet, mint a két egyedtípus közöttieknek, az ábrán látható példában az n:m változatot látjuk, mert ennél lehet a leginkább zavarbaejtő a feloldás feladata. Nem kell azonban semmi mást tenni, mint alkalmazni a közönséges n:m fokú kapcsolatoknál látott szabályt, és ekkor a 45. ábrán látható szerkezethez jutunk.



45. ábra

A példa egyébként az ún. darabjegyzék-problémára vonatkozik, amely jól ismert az iparban. A különféle gyártmányok szerelvényekből, azok esetleg alszerelvényekből, ezek alkatrészekből stb. épülnek fel hierarchikus szerkezetben, ahol tehát a hierarchia csúcsán a kész gyártmány van, az alján pedig a nyersanyag. Egy-egy cég általában több terméket gyárt és gazdaságosság szempontjából fontos, hogy ehhez olyan alkotóelemeket használjanak, amelyek több gyártmányba beépíthetők. A különféle gyártmányok alkatrész-hierarchiái tehát közös elemeket fognak tartalmazni, amelyek lehetőséget adnak a hierarchiák összekapcsolására. Így a szerkezet már nem hierarchia, hanem háló. Ha most egy általánosítást hajtunk végre, és a háló valamennyi csomópontját "alkatrész"-nek nevezzük (beleértve a késztermékét is), akkor az egyes alkatrészeket általánosító egyedtípus önmagával áll kapcsolatban, így kapjuk a 44. ábra szerinti n:m fokú rekurziót. Ezt a 45. ábra szerint feloldva a kapcsoló pontosan az alkatrészek alkotta termékszerkezetet írja le, vagyis azt mutatja meg, hogy egy alkatrész

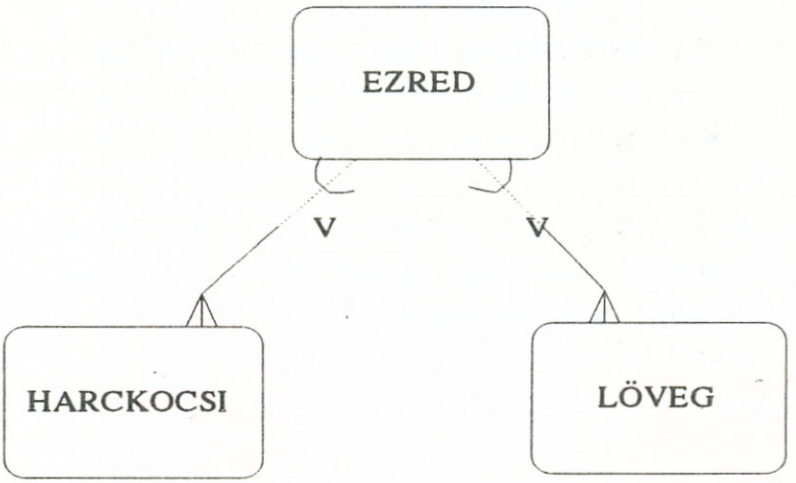
milyen más alkatrészekbe épül be (egyik kapcsolat), ill. őt milyen más alkatrészek építik fel (másik kapcsolat). Ez az, amit darabjegyzék problémának nevezünk.

A példa arra is alkalmat ad, hogy kimondjuk: a *két egyedtípus között meghatározható kapcsolatok száma nem korlátozott.* Ugyanakkor az is igaz, hogy az ilyen kapcsolatok egyrészt nem nagyon gyakoriak, számuk pedig ritkán haladja meg a négyet.

További szabály, hogy *egy egyedtípus korlátlan számú kapcsolatban vehet részt.*

Kapcsolatok nem ágaztathatók el és nem is találkozhatnak.

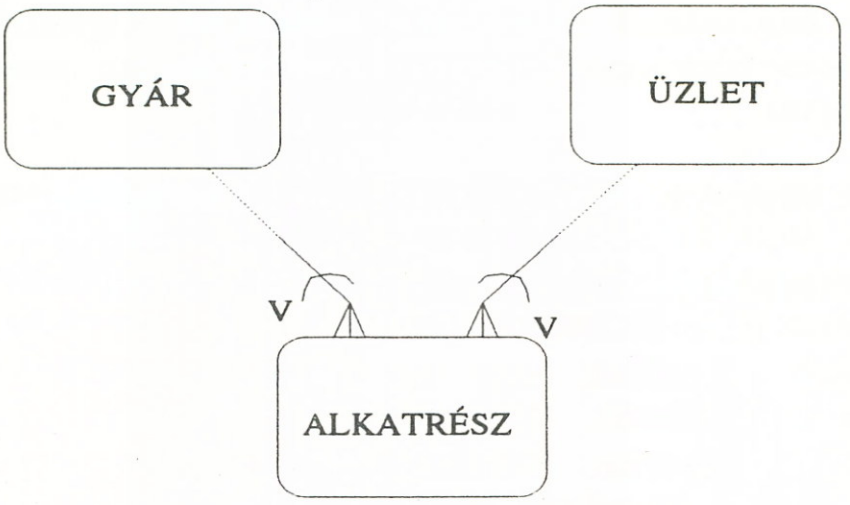
A kapcsolattípusok - amellett, hogy van fokuk és jellegük - egymás közötti viszonyukban is vizsgálhatók. Előfordulhat, hogy két, vagy több kapcsolat kölcsönösen kizárja egymást. Tételezzük fel pl., hogy egy katonai célú adatmodellben létezik a következő három egyedtípus: EZRED, HARCKOCSI, LÖVEG és az ezek által alkotott modellrész a 46. ábra szerinti.



46. ábra

Az ábrán a kapcsolatok vonalaira jelyezett ívek és a "V" betűk azt jelzik, hogy ez a két kapcsolat ún. kizáró viszonyban van egymással, vagyis egy adott ezredhez vagy csak harckocsik, vagy csak lövegek tartozhatnak (természetesen azzal a feltétellel, hogy a valóságban ez így van). Mivel a kizárás ebben az esetben az alárendelt egyedekkel kapcsolatos, ezért ez a *kizáró alárendelt* esete. A *kizáró fölérendelt* látható a 47. ábrán. Itt egy olyan modellrészletről van szó, amely az alkatrészek beszerzési forrásait követi aszerint, hogy azok gyártótól, vagy a kereskedelemből származnak. A modell azzal a feltételezéssel készült, hogy egy meghatározott alkatrész (pl. anyáscsavar) vagy csak gyártól, vagy csak üzlettől

szerezhető be. (koránt sincs persze arról szó, hogy ez a feltételezés más, hasonló alkalmazás esetében is igaz!)



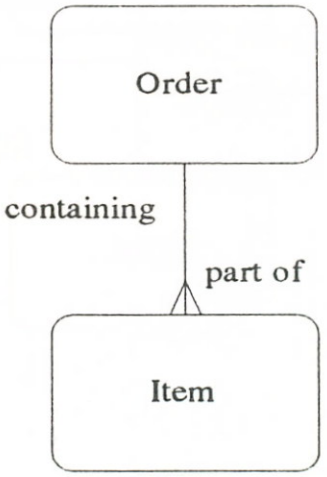
47. ábra

Amint arról fentebb már szóltam, a logikai adatmodellezés ún. egyed-kapcsolati modell megalkotását tűzi ki célul, ezért itt nem foglalkozunk az általában vett adatmodellek harmadik alapelemével, a tulajdonságtípusokkal.

Ahhoz, hogy a modell (diagram) elemeire egyértelműen hivatkozni lehessen, szükség van ezek azonosítására. Az egyedtípusok esetében ezt már lényegében meg is tettük azáltal, hogy neveket adtunk nekik. Most ezt azzal a megjegyzéssel kell kiegészíteni, hogy ezeknek a neveknek egy-egy adatmodellen belül egyedieknek kell lenniük, tehát nem ismétlődhetnek.

A kapcsolattípusoknak "ortodox" SSADM-felfogásban nem is egy, hanem két nevet kell adni, az egyiket a fölérendelt, a másikat az alárendelt egyedtípus szerinti szemléletben. Ennek a - bizonyára furcsának tűnő - megnevezési előírásnak a magyarázatát két dologban találhatjuk meg: a kommunikáció megsegítésére irányuló általános SSADM-törekvésben és az angol nyelv sajátos felépítésében. A kettős névadás azáltal segíti a kummunikációt, hogy így - az elképzelés szerint - pontosabban körülírható a modellbeli kapcsolat által leírt valóságos összefüggés. Az angol nyelv sajátossága egyúttal lehetővé teszi olyan kapcsolatnevek kiosztását, amelyeket az egyedtípusok neveivel összeolvasva többé-kevésbé kerek, értelmes angol mondat építhető fel. Nézzünk egy ilyen angol nyelvű példát (48. ábra). Ez a modellrész a következőképpen olvasandó a fölérendelt egyedtípus felől: *Each ORDER must be CONTAINING one or more ITEM* = Minden egyes RENDELÉSnek TARTALMAZnia kell egy, vagy több TÉTELt. Az alárendelt felől: *Each ITEM must be PART OF one and only one*

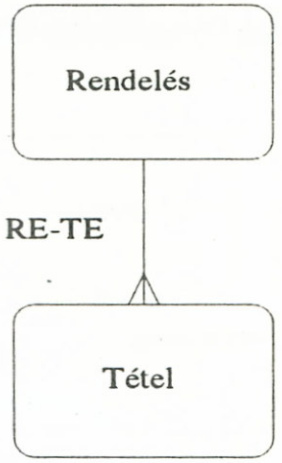
ORDER = Minden egyes TÉTELnek RÉSZEnek kell lennie egy és csakis egy RENDELÉSnek.



48. ábra

Az egyedtípusok és kapcsolattípusok angol és lehetséges magyar fordításbeli neveit a nagybetűs szedés mutatja. Látható, hogy a magyar mondatokban kénytelenek vagyunk ragokat illeszteni ezekhez a nevekhez és egy kicsit pontosabban utánagondolva arra sem nehéz rájönni, hogy magyarban - ellentétben az angollal - igen nehéz ezeknek a mondatoknak a szerkezetét szabványosítani. Ez viszont azt jelenti, hogy a kommunikáció pontosítására, javítására irányuló szándékot a magyar gyakorlatban - véleményem szerint - nem tudjuk elősegíteni kettős névadással. Van azonban itt még egy gond, amire Halassy hívta fel a figyelmemet: *elvileg* is helytelen, ha *egy* objektumnak *két azonosító* neve van.

Ez a könyv persze nem az SSADM kritikája kíván lenni, tehát ha az olvasó tartani kívánja magát a szabályokhoz, akkor ám használjon két nevet minden kapcsolattípushoz. Én a továbbiakban csak egy nevet alkalmazok egy kapcsolathoz és nem törekszem a nyelvileg kerek mondatokban való olvashatóságra, hanem a rövid és egyértelmű azonosításra. A fenti példa magyar változata ennek megfelelően pl. a következő lehet (49. ábra):



49. ábra

Nézzünk meg ezek után egy példát, amely egy kereskedelmi környezetből származó egyszerűsített modellt mutat be (50. ábra). A modell által ábrázolt valóságos világbeli környezet a következő:

Egy nagykereskedelmi vállalatnak sok vevője van szerte az országban. A vevők rendeléseket adnak fel a vállalat által forgalmazott cikkekre. Egy-egy rendelés általában több cikkre vonatkozik.

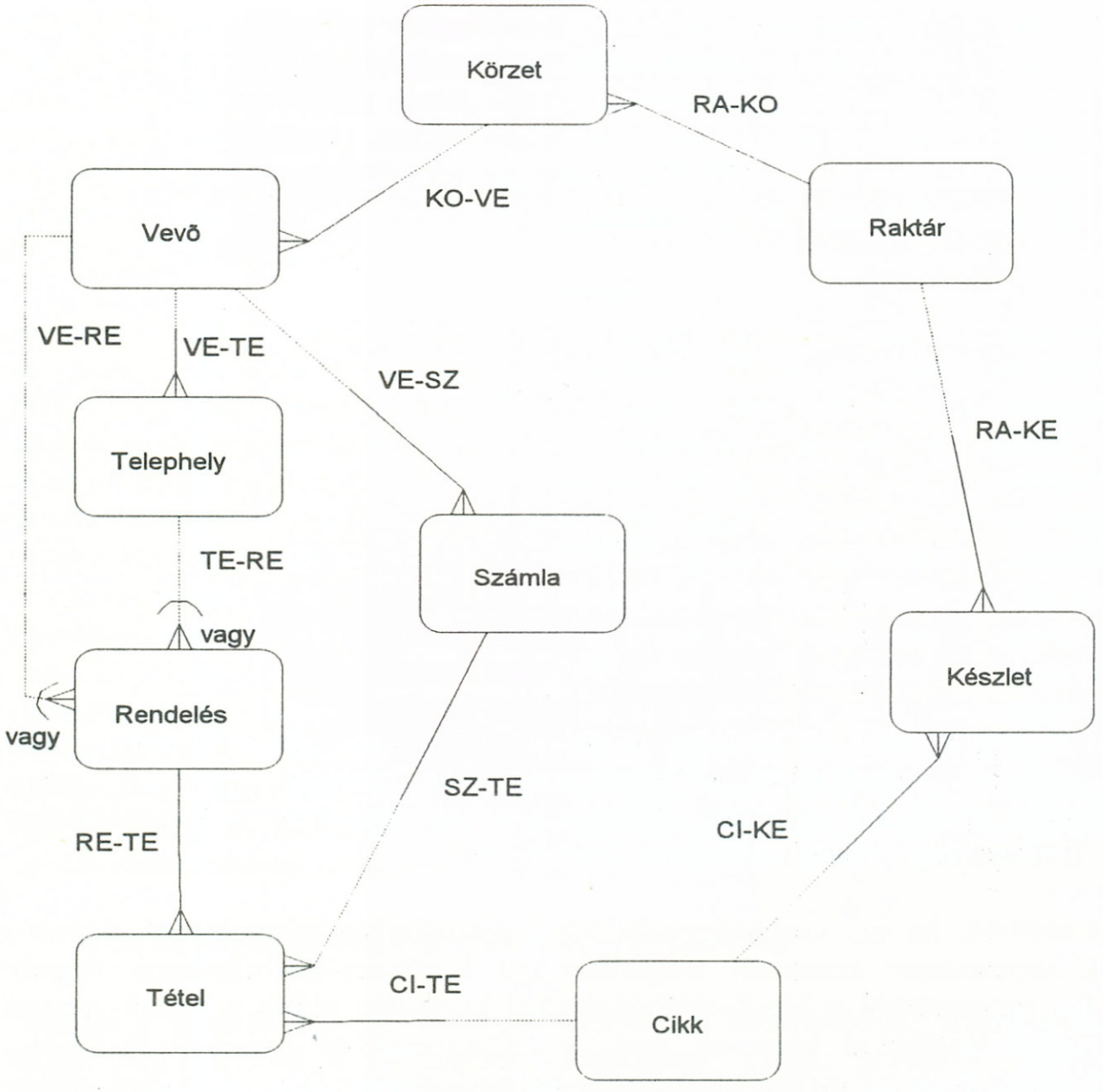
A vevők gyors kiszolgálása érdekében a vállalat körzetekre osztotta fel az országot, és minden egyes vevőt egyértelműen besorolt valamelyik körzetbe.

A cég több nagy raktárral rendelkezik szerte az országban, de egy vevőt mindig ugyanaz a meghatározott raktár szolgálja ki. Egy raktár egyébként több körzet vevőit is ellátja, vagyis a vevők területi besorolásuk alapján tartoznak valamelyik raktárhoz.

A rendelések alapján a vevők részére leszállított árúról számla készül a vevő számára.

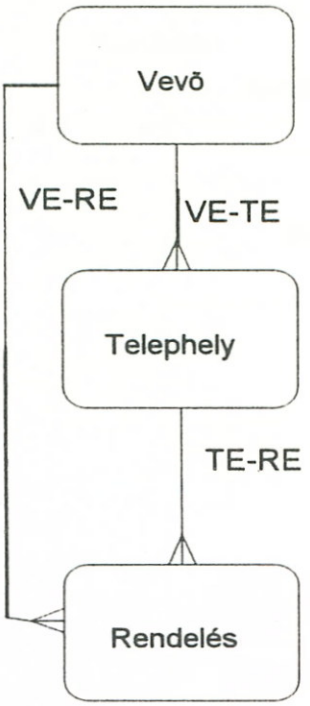
A diagramból kiolvasható - többek között - hogy a vevőknek telephelyeik is lehetnek, amelyek maguk is adhatnak fel megrendeléseket. Egy adott megrendelés azonban vagy a központtól (vevő), vagy a telephelytől származik, ezért a megfelelő két kapcsolat kizáró viszonyban van egymással.

A Vevő-Telephely-Rendelés által alkotott modellrészt érdemes külön is elemezni, mert általánosítható tapasztalatokat szűrhetünk le belőle.



50. ábra

Ez a három egyedtípus olyan szerkezetet alkot, amelyet szoktak "háromszög"-nek, vagy "tranzitív"-nek is nevezni. Érdekessége, hogy a Vevő-től elindulva két "úton" is el tudunk jutni a Rendelés-hez és feltehető az a kérdés, hogy vajon ez mennyiben indokolt. Másképpen fogalmazva: indokolja-e valami az ilyen szerkezetek létét? Most ne is a fenti modellre gondoljunk, hanem általában az olyan szerkezetekre amilyen önállóan látható az 51. ábrán. Ebben a szerkezetben a VE-RE kapcsolatot célszerűen "áthidaló" kapcsolatnak nevezhetjük, a kérdés pedig mostmár az, hogy mikor indokolt az áthidaló kapcsolatok léte, függetlenül attól, hogy hány másik egyedtípust hidalnak át.



51. ábra

Az ábrán a kapcsolatok folyamatos vonallal szerepelnek, de nem azért, mert mindkét oldalról kötelezőnek feltételezzük őket, hanem mert a jelleg előfeltételezése nélkül kívánjuk az áthidalás kérdését elemezni. A kapcsolatokat tehát itt - átmenetileg - jelleg nélkülinek feltételezzük.

Abban az esetben, ha egy meghatározott Vevő-előfordulást feltételezünk, ehhez - a VE-RE kapcsolaton keresztül meghatározott Rendelés-előfordulások fognak tartozni. Ha ugyanebből a Vevő-előfordulásból kiindulva előbb a hozzá tartozó Telephely-előfordulásokat keressük meg, majd az ezekhez kapcsolt Rendelés-előfordulásokat, és ez az utóbbi előfordulás-halmaz megegyezik az elsővel, akkor egyértelmű, hogy az áthidaló kapcsolatnak nincsen olyan szemantikai jelentése, ami nélküle is ne lenne benne a modellben. Ha ez így van, akkor pedig kimondhatjuk, hogy az áthidaló kapcsolat redundanciát visz a modellbe.

Ha viszont az előbbi két előfordulás-halmaz eltér, úgy szükség van az áthidaló kapcsolatra.

Azt, hogy melyik esetről van szó, csak a valóságos világbeli helyzet alapos elemzésével lehet eldönteni. Visszatérve a nagykereskedelmi vállalat példájára, ott indokolt volt az áthidalás megtartása, mert az egyik rendelés-halmaz a Vevő (központjához tartozó) megrendeléseket, a másik viszont a Telephelyek megrendeléseit tartalmazta.

Az áthidalás esetének a vizsgálata egy másik, általánosabb kérdéshez is elvezet. Láttuk, hogy az áthidaló kapcsolat adott esetben redundáns lehet. No de miért baj ez? - kérdezhetné valaki. Azért, mert a logikai adatmodellezés során olyan modellt szeretnénk elkészíteni, amely *optimális, ami azt jelenti* - nagyon egyszerűen fogalmazva - *hogy minden benne van, ami kell, és csak az van benne.* Ennek az elvnek a teljeskörű kifejtése nem lehet ennek a könyvnek a feladata, de röviden a következőkről van szó:

Az adatmodellezés az adatbáziskezelési technikát feltételező adatbázis tervezésből fejlődött ki. Az adatbáziskezelést megelőzően a fájlok szervezését az őket használó programok igényeihez igyekeztünk igazítani, ezzel szemben az adatmodellezés az adatok feldolgozásoktól független belső logikáját igyekszik megragadni, mert arra számít, hogy nagyon sokféle feldolgozás adatigényeit kell kielégítenie, tehát a tárolás szerkezetét egyikhez sem lehet igazítani a másik rovására. Ugyanakkor biztosítani kell, hogy az adatok bármilyen - előre nem is látható - kombinációban is visszakereshetők legyenek a modellből, ami csak úgy lehetséges, ha akármelyik egyedtípusnál elkezdhetjük a keresést és onnan bármelyik másik egyedtípushoz el tudunk jutni a kapcsolatokon keresztül az azokban szereplő adatok (tulajdonságok) elérése céljából. Ez annyit is jelent, hogy a modellben nem lehet szakadás, de - ha a feladatot optimálisan akarjuk megoldani akkor - felesleges (redundáns) kapcsolat sem. Ilyen felesleges kapcsolat az áthidalásoknak az a típusa, amely az áthidalás nélkülivel azonos eredményre vezet. Az optimalitásnak egyéb vonatkozásai is vannak.[8]

10.3 A logikai adatmodellezés gyakorlata

Miután a logikai adatszerkezetek alapelemeit áttekintettük, foglalkozzunk most azzal, hogyan kell a modelleket elkészíteni. Ehhez példaként egy könyvtári környezetet fogunk felhasználni, melynek rövid leírása a következő:

Egy iskolai könyvtárból a tagok könyveket kölcsönözhetnek. A könyveket egy vagy több író írta, és természetesen az egyes íróktól lehet többféle mű is a könyvtár birtokában. A népszerű könyvekből több példányt is tart a könyvtár.

Egy kölcsönző egyszerre legfeljebb hat könyvet kölcsönözhet. Ha a könyveket határidőre nem hozzák vissza a kölcsönzők, akkor a könyvtár

[8] A téma iránt érdeklődőknek javaslom dr. Halassy Béla adatmodellezési tárgyú könyveinek tanulmányozását.

pénzbüntetést szab ki. Amennyiben a könyveket a kölcsönző a harmadik figyelmeztetés után egy héten belül nem viszi vissza és a kiszabott büntetést nem fizeti ki, akkor "fekete listára" kerül, amíg a helyzetet nem rendezi.

Ha egy igényelt könyvből pillanatnyilag egy példány sincs bent, akkor a kölcsönző számára foglalást lehet készíteni és a könyv rendelkezésre állásakor értesítést kap.

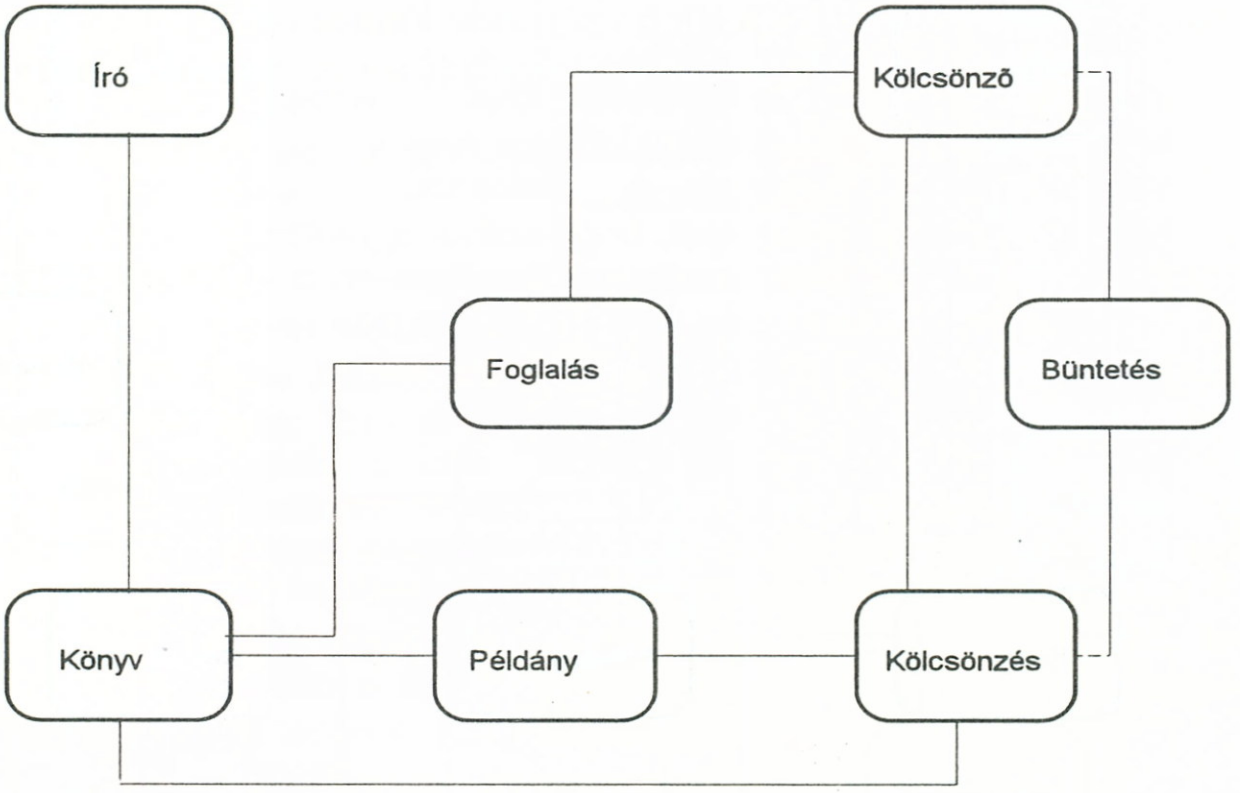
A logikai adatmodell kialakításának a lépései a következők:

1. Kezdő egyedtípusok kiválasztása

   Először a figyelembe veendő egyedtípusokat kell meghatározni az adott környezetről rendelkezésre álló információ alapján. Azért nevezzük ezeket az egyedtípusokat "kezdő"-nek, vagy "induló"-nak, mert tisztában vagyunk azzal, hogy a modell végleges alakjának eléréséig a benne szereplő egyedtípusok köre még elég sokat fog változni. Esetünkben az induló egyedtípus-jegyzéket a példaleírás alapján készíthetjük el:

   KÖNYV
   ÍRÓ
   PÉLDÁNY
   KÖLCSÖNZŐ
   FOGLALÁS
   KÖLCSÖNZÉS
   BÜNTETÉS

2. Közvetlen kapcsolatok meghatározása egy kezdődiagram megrajzolásával.
   Első közelítésben olyan "modellt" készítünk, amelyben mindössze azt tüntetjük fel, hogy a fenti egyedtípusok közül melyek hozhatók egymással *közvetlen* kapcsolatba (52. ábra).



52. ábra

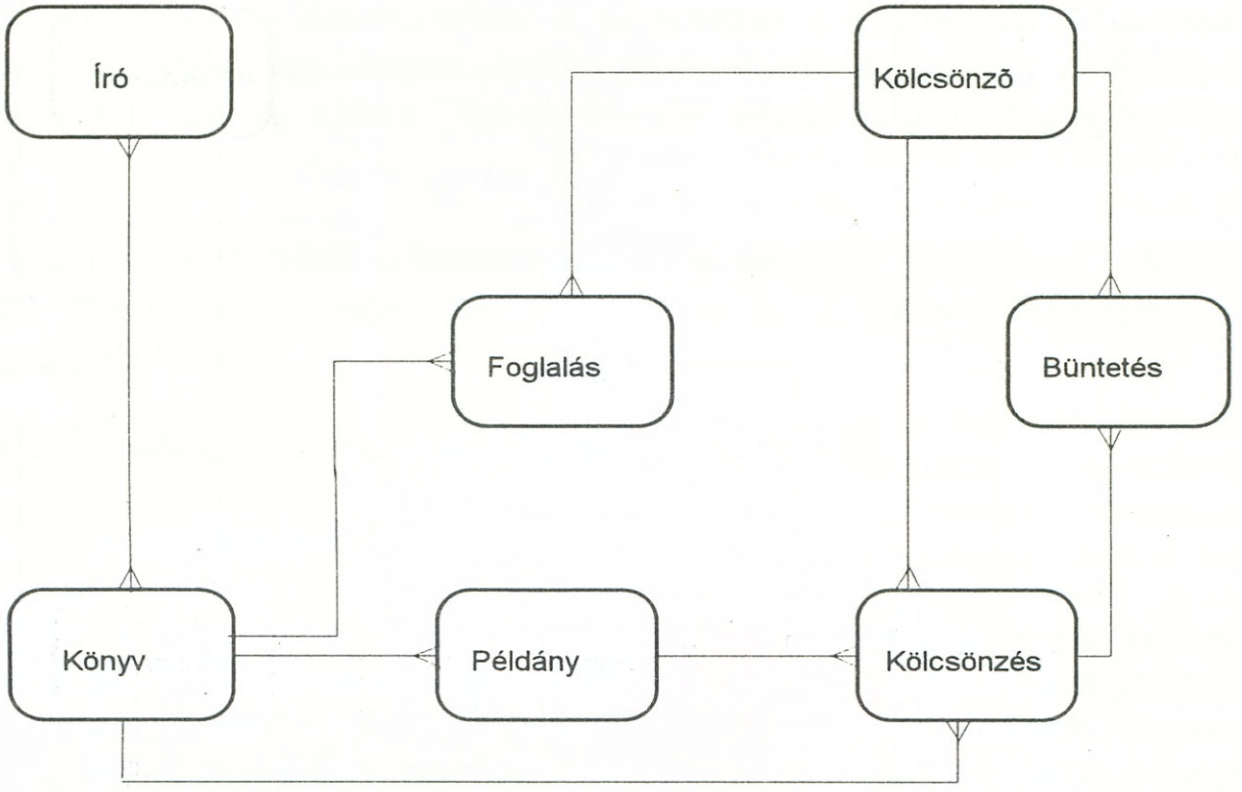
3. A kapcsolattípusok fokának meghatározása

A kiinduló, fok nélküli kapcsolatokra most rátesszük a fokot jelző "seprűket" (53. ábra).

4. Az n:m fokú kapcsolatok feloldása

Ilyenből itt csak egy van, ezt fel is oldhatjuk, de egyúttal gondolkodjunk el azon, hogy vajon tudunk-e valamiféle értelmet tulajdonítani ennek a kapcsolatnak az adott környezetben, azon kívül, hogy betölti a kapcsoló szerepkört.

Szemmel láthatóan ez az egyedtípus "hozza össze" egymással az írót és a könyvet, tehát ez mutatja meg, hogy melyik könyvnek melyik



53. ábra

író a szerzője. Adjuk ezért ennek az egyedtípusnak azt a nevet, hogy SZERZŐ. A kapott eredményt az 54. ábra mutatja.

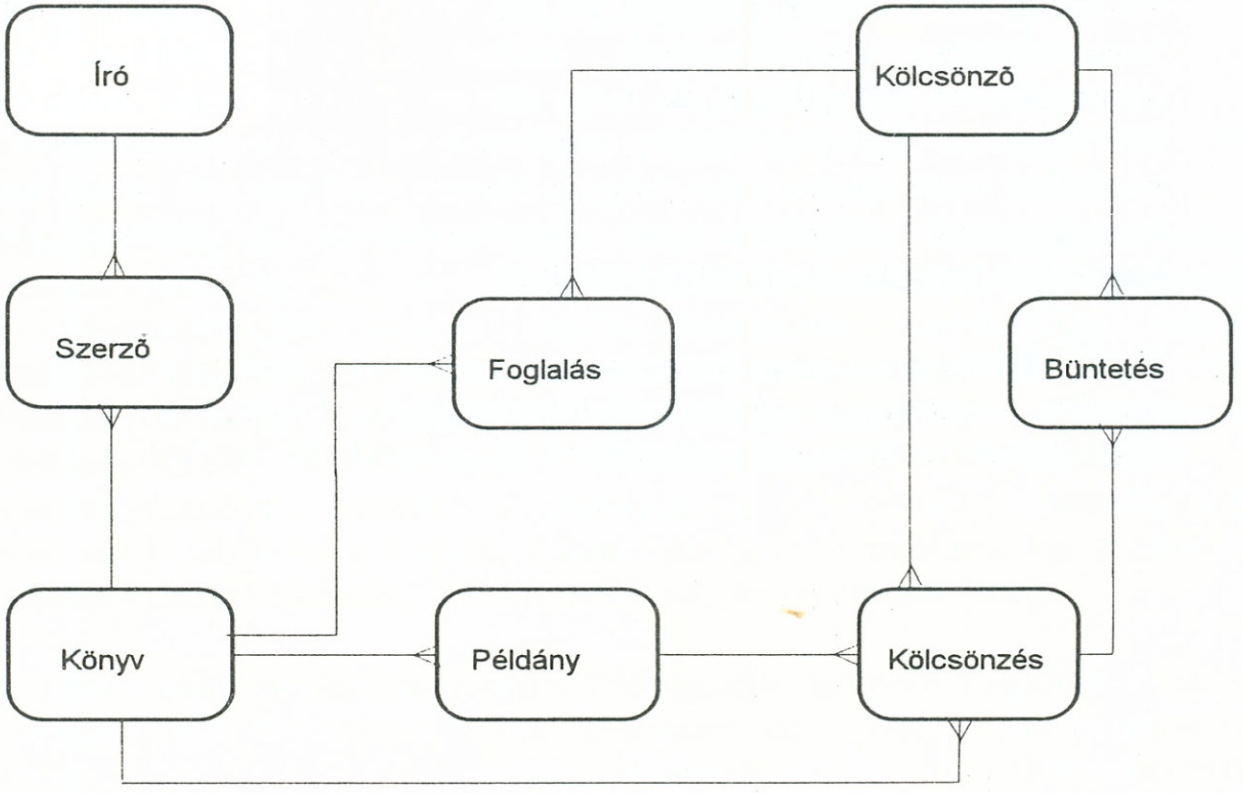
5. A redundáns kapcsolatok eltávolítása

Könnyű felfedezni, hogy a KÖNYV, a PÉLDÁNY és a KÖLCSÖNZÉS egyedtípusok közötti kapcsolatok közül a KÖNYV és a KÖLCSÖNZÉS közötti ún. áthidaló kapcsolat. Amennyiben elvégezzük a fentebb ismertetett elemzését, azt az eredményt kapjuk, hogy ez a kapcsolat elhagyandó a modellből.

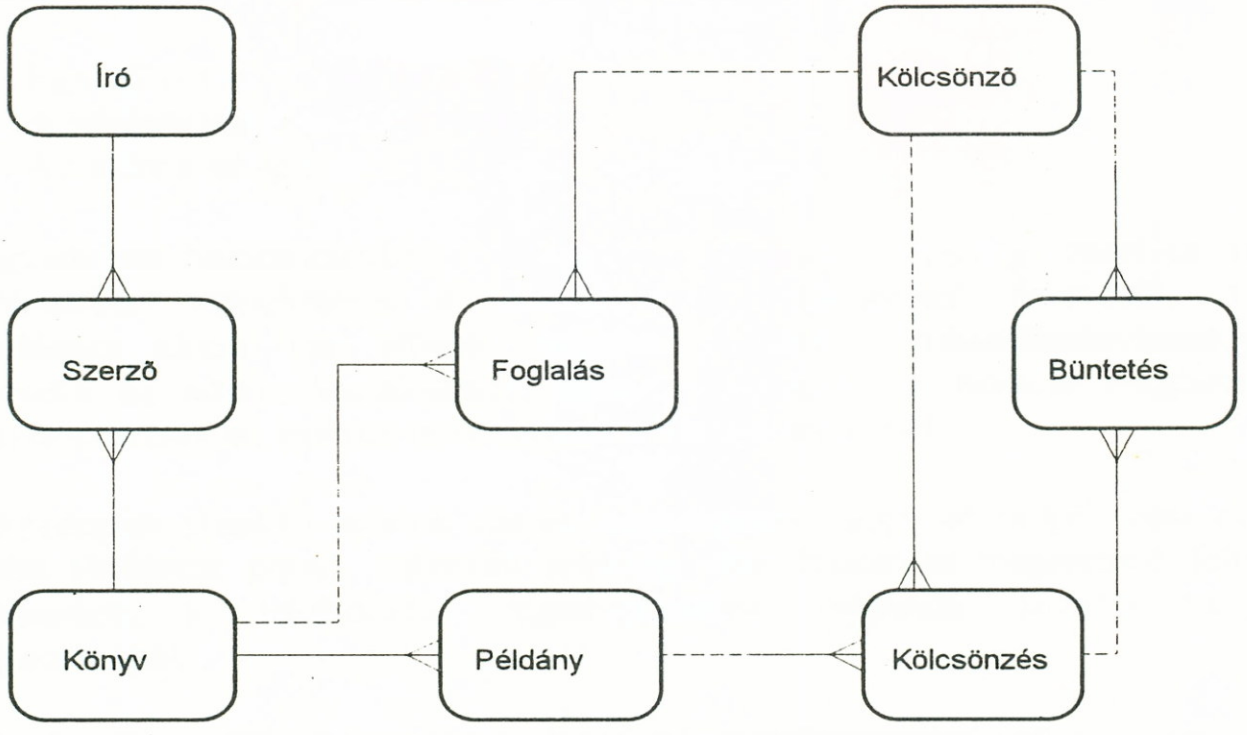
6. A kapcsolatok jellegének és egymáshoz való viszonyának elemzése

Ez jelenti egyrészt a kapcsolatok kötelező, vagy esetleges jellegének megvizsgálását minden kapcsolat esetében mind a fölé-, mind pedig az alárendelt szempontjából, valamint a kapcsolatok esetleges kizáró viszonyának feltárását és a diagramon való feltüntetését.

Az 5. és 6. pont szerinti elemzési eredmények átvezetése a modellen elvezet ahhoz a végeredményhez, amely az 55. ábrán látható.



54. ábra



55. ábra

7. A modell ellenőrzése

Az ellenőrzés a felhasználó által megfogalmazott visszakeresési követelmények szerint történik. Ez azt jelenti, hogy a legfontosabbnak ítélt visszakeresési követelmények kielégítését végigkövetjük a logikai adatmodellen. Megállapítjuk, hogy melyik egyedtípus az, ahol a keresést el kell kezdeni (belépési pont), majd milyen más egyedtípusokhoz kell a modellen belül eljutni azért, hogy a keresés által igényelt valamennyi adatot "összeszedjük". Ezekhez az egyedtípusokhoz a belépési ponttól a kapcsolattípusokon keresztül jutunk el.

Ez az ellenőrzés a visszakeresési utak modellezésének témaköréhez tartozik, amelyet - minthogy a lekérdező feldolgozások tervezésében kiemelt szerepe lesz - külön technikaként tárgyalunk (ld. következő fejezet).

11. fejezet
A visszakeresési út modellezése

Ez a technika nagyon szorosan kapcsolódik a logikai adatmodellezéshez. Jelentőségét az adja meg, hogy az SSADM már a követelmények nyilvántartásba vételének megkezdésétől kezdve külön csoportba sorolja az adatok visszakeresésével (lekérdezés), ill. karbantartásával kapcsolatos igényeket, majd pedig a rendszertervezésnél is más technikákat rendel hozzájuk. A visszakeresési típusú feldolgozások tervezésének megalapozása a visszakeresési út modellezésénél kezdődik. Emellett a technika azt a szerepet is betölti, hogy segítségével válik lehetővé a logikai adatmodell első ellenőrzése (a második a relációs adatelemzéssel történik, ami a következő fejezet témája lesz).

A logikai adatmodell az adatok minden feldolgozási igénytől függetlenül létező, belső szerkezetét tükrözi vissza. Ilyen szerkezet egy információs rendszerben természetesen csak egy van, ugyanakkor nagyon sokféle visszakeresési igény lehetséges, ami szintén természetes.

A modellezést a következő lépésekben hajtjuk végre:

- Egyedi nevek rendelése a lekérdezésekhez.
- A lekérdezést elindító adatok meghatározása.
- Az elérési út specifikálása.

Az egyedi név hozzárendelése a visszakereséshez azért lényeges, mert ez fogja a továbbiakban végigkísérni a visszakeresési feldolgozás tervezését. A név megadására akkor van először módunk, amikor a követelménykatalógusba felvesszük az adott visszakeresési igényt. Legkésőbb a funkció meghatározás idején dönthetjük el, milyen nevet adunk a visszakeresésnek.

A lekérdezést elindító adatok ahhoz szükségesek, hogy el tudjuk érni az első egyedet (belépési pont), valamint szükség lehet bizonyos megszorító feltételek megadására a lekérdezés végrehajtásakor elérendő további egyedek kiválasztásánál.

Az elérési út specifikálása során a következő tevékenységeket kell elvégezni:

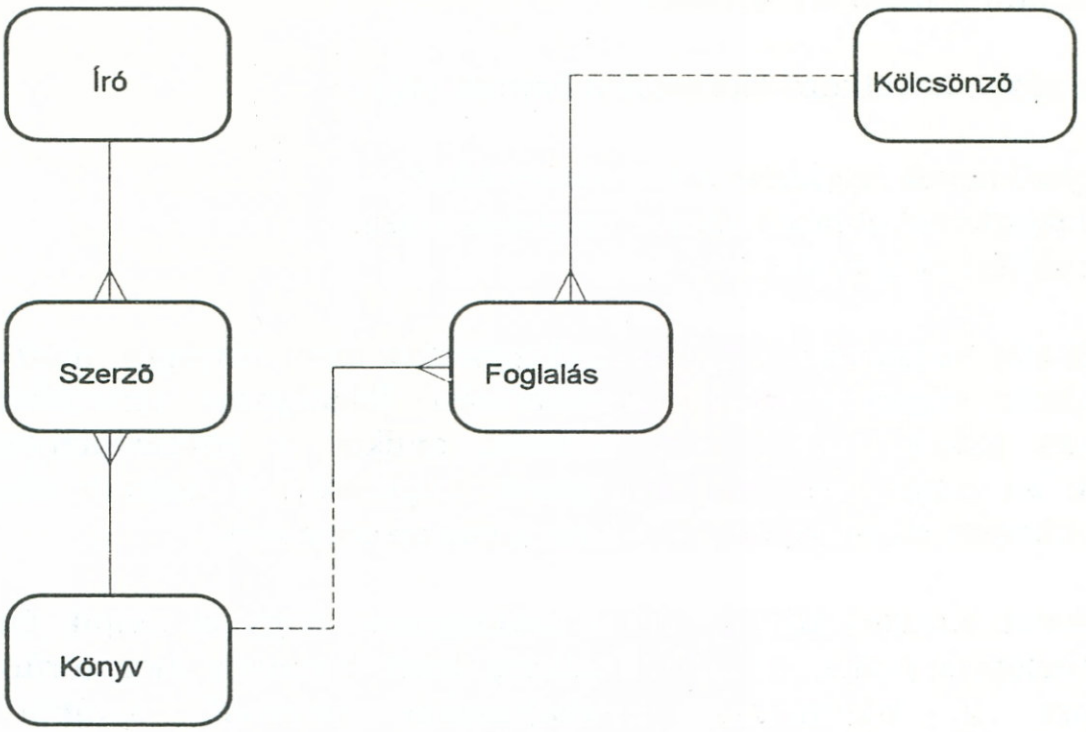
1. meg kell határozni a logikai adatmodellből azokat az egyedeket, amelyekre a lekérdezéshez szükség lesz, ez lesz *a lekérdezés adat-almodellje*
2. el kell dönteni, hogy ezek közül melyiknél kezdjük el a visszakeresést, ez lesz a *belépési pont*
3. meg kell határozni, hogy a belépési ponttól kiindulva milyen útvonalon járjuk végig az adat-almodellt, ez lesz *a lekérdezés navigációs útja.*
4. végül a navigációs utat Jackson-szerkezetek használatával kell leírni

Kövessük végig a lekérdezés modellezését az előző fejezetben meghatározott könyvtári adatmodellen. Példaképpen válasszuk azt a lekérdezést, melynek során *vissza akarjuk keresni mindazokat a kölcsönzőket, akik egy meghatározott író műveire várakoznak.*

A szükséges egyedek meghatározásakor két igen kézenfekvő tényből indulhatunk ki:

- a KÖLCSÖNZŐ egyedtípus biztosan kelleni fog, hiszen a visszakeresendő adatok ott vannak
- az is biztos, hogy az ÍRÓ kell, ez a visszakeresési igényből világos.

Ennek alapján az érintett almodell a következő (56. ábra):

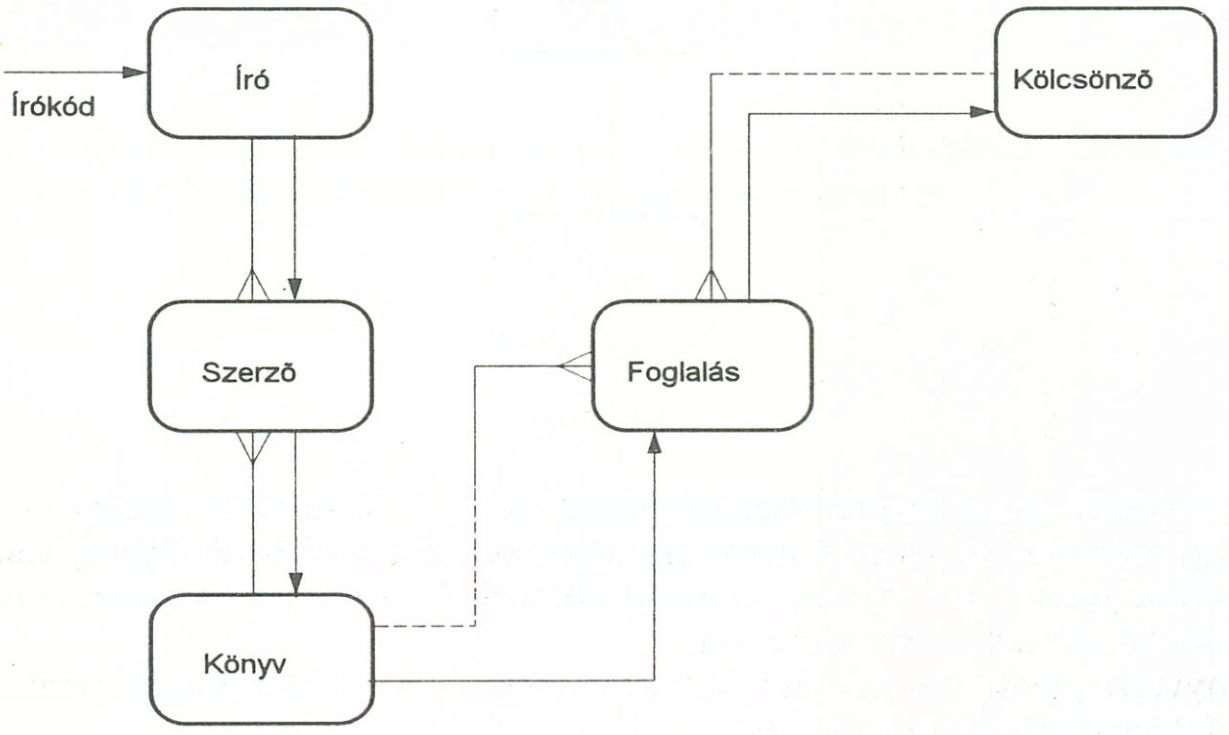


56. ábra

Ez tehát a lekérdezésünk adat-almodellje. Nem nehéz annak az eldöntése, hogy melyik egyedtípusnál kell belépnünk ebbe az almodellbe, hiszen *egy*

meghatározott író műveire várakozó kölcsönzőket akarunk visszakeresni. Más szóval pontosan behatárolható az ÍRÓ egyedtípusnak az az előfordulása, amellyel a foglalási kapcsolatban levő kölcsönzőket keressük. Az ÍRÓ lesz tehát a belépési pontunk. Ezzel kapcsolatban meg is jegyezhetjük, hogy ha a visszakeresési igény megfogalmazásában olyan kifejezés szerepel, mint "egy adott", vagy "egy meghatározott", "kiválasztott", akkor az az egyedtípus lesz a belépési pont, amelyre az ilyen kifejezés vonatkozik. Ilyenkor azt feltételezzük, hogy a belépési pont elérése az adott egyed elsődleges kulcsa segítségével történik. Megjegyezni kívánom, hogy ez az elsődleges kulcs nem biztos, hogy a felhasználó számára ismert a későbbi fizikai megvalósításban (mert mondjuk ablakból választja a belépési pontot), de ez a tervezés logikai szintjén ne befolyásoljon bennünket.

Az ÍRÓ-tól a KÖLCSÖNZŐ-ig terjedő visszakeresési utat nyilakkal jelöljük ki (57. ábra).



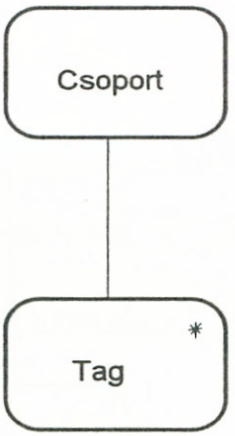
57. ábra

A belépési pontot jelző nyíl mellett feltüntettük a belépéshez szükséges tulajdonságtípust, amely az ÍRÓ azonosítója.

Végül előállítjuk a lekérdezési út modelljét az ú.n. Jackson-féle alapszerkezetek felhasználásával. Ez annyit jelent, hogy az elérési nyilakat annak megfelelően használjuk, hogy egy egyedelőfordulástól egy vagy több más típusú egyedelőfordulást kell-e elérni. Ez alapvetően összefügg azzal, hogy a

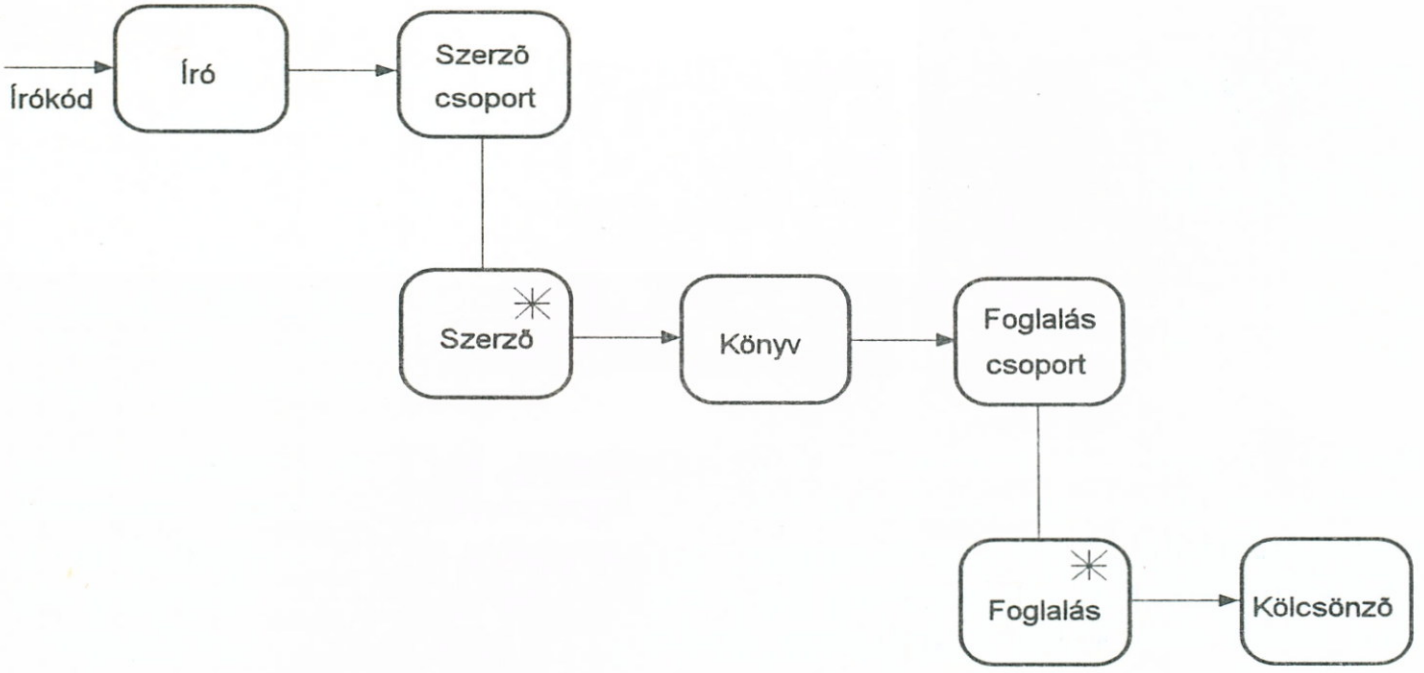
kapcsolatokon milyen irányban haladunk át a lekérdezés teljesítéséhez szükséges navigáció során.

Ha az alárendelttől a fölérendelt felé haladunk, akkor egy alárendelt-előforduláshoz mindig csak egy fölérendelt-előforduláshoz juthatunk. Ellenkező irányban haladva, egy fölérendelt-előforduláshoz több alárendelt-előfordulás tartozhat (legalább is általában és külön megszorítások híján). Ezt úgy is értelmezhetjük, hogy a navigáció egy lépésekor - amely megfelel egy egyedtípustól egy másikhoz való továbbhaladásnak - vagy egy egyedelőforduláshoz, vagy egyedelőfordulások egy csoportjához jutunk. A továbbiakban a csoportot külön jelöljük, amelyen belül a csoport egyes tagjainak elérését jacksoni értelemben vett iterációnak tekintjük (58. ábra).



58. ábra

A tag esetében ennek megfelelően csillaggal jelezzük az iterációt. Magát az elérési utat úgy rajzoljuk át, hogy amikor egy lépés során egy előforduláshoz, vagy egy csoporthoz jutunk, azt vízszintes nyíllal ábrázoljuk, a csoporthoz tartozó tagokat pedig az 58. ábra szerinti iterációval.
A könyvtári példa szerinti visszakeresést ezeknek a jelölési megállapodásoknak megfelelően ábrázolja az 59. ábra.



59. ábra

Az így kapott diagramot tekintjük az elérési út modelljének, amelyet majd felhasználunk a logikai adatbázis feldolgozások tervezésénél.

12. fejezet
Funkciómeghatározás

Amikor a rendszer fejlesztése során megtörtént a kívánt rendszer logikai adatfolyam diagramjainak kidolgozása a legrészletesebb szintig, másrészről rendelkezésre áll a kívánt rendszer logikai adatmodellje, akkor elérkezett az idő az összegyűlt ismeretek rendezéséhez. Ezt a feladatot tölti be a funkciómeghatározás, amit sokan nem külön technikaként értelmeznek, hanem - amint említettem - meglevő ismeretek rendezéseként. Ezek az ismeretek a tervezett rendszerre vonatkoznak. A meglevő rendszerre (ha van ilyen) nem készítünk funkciómeghatározást.

Ennek során megtörténik azoknak az alapvető feldolgozási egységeknek a meghatározása, amelyekre azután ráépül a rendszer további tervezése. A feldolgozási egységek, vagy más néven funkciók az inputokon és outputokon keresztül kapcsolódnak a környezetükhöz, meghatározásuk tehát ezeket is magában foglalja.

Ezen túlmenően a funkciómeghatározás kapcsolatot teremt az SSADM 3. szakaszában készülő két termékcsoport között, amelyek

- a kívánt rendszer logikai AFD-i és
- az esemény-hatás diagramok (ld. 13. fejezet).

Ennek a technikának az alkalmazása során is nagyon fontos a felhasználó-kapcsolat, mert a funkciókból a későbbiek során kialakuló feldolgozások akkor tesznek eleget a minőségi követelményeknek, ha a lehető legjobban segítik a felhasználók munkavégzését.

12.1 Mi a funkció?

A funkció az a feldolgozás, amit a felhasználó meghatározott igényének kielégítése érdekében mint egyetlen egységet kell tudni végrehajtani. Hozzá tartozik a felhasználó számára adandó eredmény, tehát az output, magának a feldolgozásnak a leírása - amely lehet lekérdezés, vagy karbantartás - és az ennek elvégzéséhez szükséges input.

A funkció és a tervezett rendszer AFD-i között az az összefüggés, hogy egy funkció legtöbbször megfelel egy elemi szintű feldolgozásnak (folyamatnak), de olykor több elemi folyamatot is magába foglalhat, vagy pedig egy elemi folyamatnak csak egy részét valósítja meg. A konkrét megoldást az dönti el, hogy melyik változat szolgálja a legjobban a felhasználó munkavégzését. Ebből a megállapításból az is következik, hogy a funkciókban a rendszerről alkotott felhasználói szemlélet testesül meg, nem pedig rendszerszervezői, vagy programozói szemlélet.

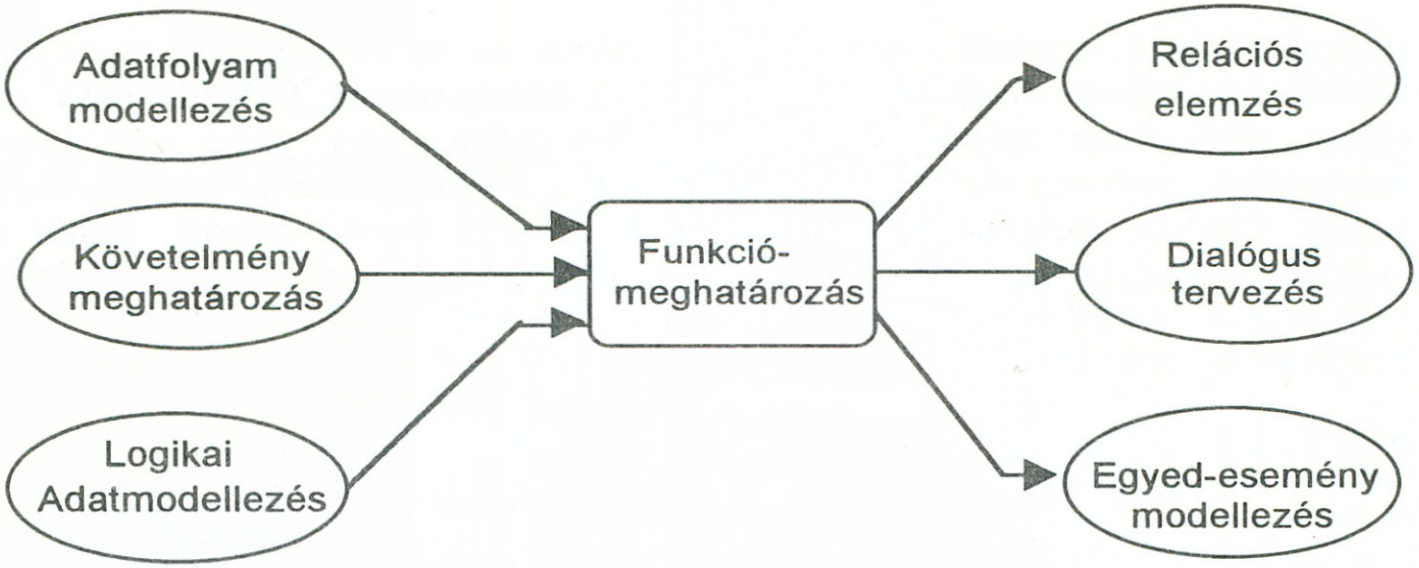
A funkciók többféleképpen csoportosíthatók:

- *A feldolgozás típusa szerint* vannak visszakeresési és karbantartási funkciók.
- *A megvalósítás módja szerint* lehet a funkció on-line, vagy off-line. Vannak esetek, amikor egy funkció mindkét módon végrehajtható.
- *A kezdeményező szerint* vannak felhasználói, ill. rendszerfunkciók

12.2 A funkciómeghatározás helye az SSADM-ben

A funkciómeghatározás - az SSADM 3. szakaszának részeként központi szerepet tölt be: összefogja az addigi tervezési elemeket és kapcsolatot teremt közöttük, valamint megnyitja az utat egy sereg további tervezési tevékenység, ill. technika alkalmazása előtt.

Összekapcsolja a tervezett rendszerre vonatkozó logikai adatfolyam modellezés, logikai adatmodellezés, valamint a követelmény meghatározás eredményeit és kiegészíti ezeket bizonyos - elsősorban az inputokra és outputokra - vonatkozó információval. Az I/O szerkezet meghatározása lehetővé teszi az adatmodellezés pontosítását a relációs elemzés végrehajtásával, a dialógusok és menük megtervezését, valamint az egyed-esemény modellezés végrehajtását (60. ábra.)



60. ábra

A dialógustervezést követheti a menürendszer megtervezése, az egyed-eseménymodellezés pedig majd lehetővé teszi az ún. logikai adatbázis-feldolgozások közül a karbantartó feldolgozások logikai tervezését. A lekérdező feldolgozások logikai tervezését a relációs elemzéssel is ellenőrzött adatmodellen értelmezett visszakeresési út modellek és az I/O szerkezet teszi lehetővé.

12.3 A funkciómeghatározás eredménye

Minthogy itt az elkészülő termék nem diagram, megismerése azáltal történhet, hogy áttekintjük a funkcióleíró formanyomtatványnak (ld. következő oldalon) a meghatározás szempontjából érdeminek tekinthető rovatait, és értelmezzük a tartalmukat.

A *funkció típus* rovatban adjuk meg, hogy karbantartó/lekérdező, on-line/off-line, ill. felhasználói vagy rendszer-funkcióról van szó (ld. 12.1 pontban).

A *felhasználói szerepkörök* rovatában on-line funkció esetében meg kell adni azokat a felhasználócsoportokat, amelyeknek jogosultságuk lesz az adott funkció használatára. Azért nevezzük ezeket szerepköröknek, mert adott esetben igen különböző beosztásban dolgozó felhasználók lehetnek érdekeltek a funkció használatában, tehát magának a funkciónak a használata egyfajta közös szerepkörbe hozza össze ezeket a felhasználókat. A szerepkörök és funkciók kapcsolatának ismerete nemcsak a jogosultságok kezelésének megtervezése szempontjából fontos, hanem a menüszerkezet megtervezésekor is figyelembe kell venni. A menüket és dialógusokat ugyanis - amelyeknek a "végéhez csatlakoznak" a funkciók - a felhasználói szerepekhez igazítjuk.

A *funkció leírása* röviden tartalmazza, hogy mi váltja ki a funkció elindítását, mi a rendszer válasza az elindító inputra, és hogy ez milyen outputokban jelentkezik. Ha létezik a funkció megjelenítésének módjára utaló felhasználói igény, akkor ezt itt kell rögzíteni, mert ez az információ jól felhasználható lesz a dialógusok tervezésénél.

A funkcióval összefüggésben már ezen a szinten tudomásunkra juthat *hibakezelés*i jellegű információ. Ha így van, akkor ezt foglaljuk össze a megfelelő rovatban, de semmiképpen sem kell teljességre törekednünk, mert ez majd a fizikai tervezés feladata lesz.

Nagyon lényeges a funkciók és adatfolyam diagramok közötti kapcsolat, mert a funkciómeghatározás kiindulópontja - legalábbis a karbantartó funkciók esetében - a tervezés alatt álló rendszer logikai AFD-je. Ennek a kapcsolatnak a jelölésére szolgál a formanyomtatvány *AFD folyamatok* c. rovata. A legtöbb esetben az AFD elemi szintü folyamatai közvetlenül (1:1) funkcióvá válnak. Előfordulhat azonban az is, hogy több folyamatból lesz egy funkció, vagy éppen ennek az ellenkezője.

Több folyamatból általában akkor lesz egy funkció, ha egyrészt az elemi folyamatok valóban elemiek (tehát igen egyszerűek), másrészt az AFD-n közvetlen folyamat-folyamat adatfolyam van (61. ábra)

Ha ilyen folyamat-összevonás történik, akkor a funkcióleíró formanyomtatványon mindegyik összevont folyamatra hivatkozni kell.

Ennek az ellenkezője általában akkor fordul elő, ha az AFD-n a folyamatok elemi szintjét viszonylag "magasan" (nem túl részletesen) határozzuk meg és kénytelenek vagyunk ezt a "hiányosságot" a funkciómeghatározás során mintegy bepótolni. Ilyenkor az AFD-n még egyetlen folyamatot több funkcióban valósítjuk meg, tehát az ugyanarra az AFD-folyamatra való hivatkozás több funkció formanyomtatványán fog megjelenni.

FUNKCIÓ MEGHATÁROZÁS			SSADM-4		
Projekt/rsz.	Elemző	Dátum	Változat	Állapot	Oldal
BAF	*Dr. Bana*	*93.6.29.*	*V1*	*M*	*1/1*

Funkciónév		Funkció azonosító	
Típus			
Felhaszn. szerepek			
Funkcióleírás			
Hibakezelés			
AFD eljárások			
Események		Esemény gyakoriság	
I/O leírások			
I/O szerkezetek			
Követelménykatalógusra hivatkozás			
Tömegszerűség			
Kapcsolódó funkciók			
Lekérdezések		Lekérdezés gyakor.	
Közös feldolgozás			
Dialógusnevek			
Szolgáltatások szintjére vonatkozó körülmények			
Leírás	Célérték	Tűrés	Megjegyzés

61. ábra

Magának a funkciónak a leírása mellett a meghatározás hasonlóan fontos eleme az inputok és outputok megadása, ezért találunk a formanyomtatványon két hivatkozási rovatot is erre a célra. Az egyik, az *I/O leírások* rovat az adatfolyam

diagramok felé teremt egy másik kapcsolatot. Az adatfolyamok közül azokat, amelyek átlépik a rendszer határát, inputoknak, ill. outputoknak tekintjük, és külön is dokumentáljuk. A rájuk való hivatkozás a kapcsalotos környezeti elem azonosítójának és a megfelelő AFD-beli elemi folyamat azonosítójának kötőjellel való összekapcsolásával történik. Ha a funkció több elemi folyamat összevonásával született, akkor az ezekkel kapcsolatos valamennyi eredeti I/O-ra a fent leírt módon kell hivatkozni.

Az inputok és outputok tartalmi felépítésének egy pontosabb megadását foglalják magukba az ún. *I/O szerkezet* diagramok és kapcsolódó leírásaik. Ezekre hivatkozunk az I/O szerkezetek rovatában. Az I/O szerkezetek kialakítása egyébként a funkciómeghatározás résztechnikája, és mint ilyenre még ebben a fejezetben visszatérünk.

A funkciókkal kapcsolatban azt is meg kell adni, hogy mi vezetett a megjelenésükhöz, vagyis hogy melyik felhasználói igény kielégítésével kapcsolatban lett szükség rájuk. Ezt kell feltüntetni a funkciómeghatározás formanyomtatványán a *Követelménykatalógusra hivatkozás* c. rovatban.

A funkciók meghatározása alapozza meg a feldolgozások logikai tervezését, amelyeket majd fizikai szinten is meg kell tudni tervezni, ehhez pedig szükség lesz annak a technikai (hardver/szoftver) környezetnek a kiválasztására, amelyre a fizikai terv készül. Ismerni kell tehát a logikai tervezési szint által támasztott adattárolási, ill. feldolgozási követelményeket (ezeket nevezzük a szolgáltatási szintre vonatkozó követelményeknek), aminek alapján becsülhető, hogy milyen tárolási és feldolgozási kapacitásokat kell az adott információs rendszer üzemeltetéséhez biztosítanunk. Ezeket a követelményeket a különféle dokumentumokon általában az ún. *mennyiségi adatok* adják meg. Találunk ilyet a funkciómeghatározás dokumentumán is és ide azt kell beírni, hogy időegység alatt (amit megválaszthatunk) átlagban hányszor kerül sorra a funkció végrehajtása. Célszerű az átlagtól való jelentős eltérések jelzése is, ha tudomásunk van ezek várható bekövetkezéséről és ha számszerűsíteni is tudjuk az eltéréseket.

Az SSADM erőssége - és egyben talán gyengesége is - az erőteljes kereszthivatkozási rendszer, amely igyekszik a lehető legszorosabban követni a tervezési objektumok egymáshoz való kapcsolódását. Lehetnek olyan funkciók közötti összefüggések, amelyek esetleg nem követhetők az adatfolyam diagramokon, de mégis lényegesek. Ezek feltüntetésére van lehetőség a *Kapcsolódó funkciók* rovatában. Ilyen lehet pl. egy lekérdezési funkcióra való hivatkozás egy karbantartási funkció meghatározásában, ahol a karbantartás során mondjuk létrejön egy napló-fájl, amelyet egy másik funkció keretében le lehet kérdezni.

A lekérdezési funkciók esetében mindig van egy visszakeresési út, amelyre szükség van a lekérdezéshez, de egyes karbantartó funkciók is szükségessé tesznek lekérdezést (pl. ahhoz, hogy eldönthessük, karban kell-e tartanunk egy adatot, előbb lekérdezzük). A lekérdezések visszakeresési útjára való hivatkozást írjuk a *Lekérdezés* c. rovatba. A funkció fentebb tárgyalt gyakoriságához hasonlóan megadjuk a *lekérdezés gyakoriság*át a funkción belül.

Lehetnek egy információs rendszerben olyan elemi feldolgozások, amelyeket a rendszer különböző részeiben visszatérő jelleggel, többször is felhasználunk. Célszerű, ha ezeket csak egyszer írjuk le, aztán pedig csak hivatkozunk rájuk. Olyanok ezek, mint a programozás szubrutinjai, vagy egy programnyelvben a felhasználói programozó által előállított függvények. Itt *közös*, vagy *közhasznú feldolgozás*nak nevezzük az ilyen feldolgozásokat és hivatkozunk rájok mindazon funkciókban, amelyek használják őket. Ezeknek egyébként meg kell lennie a pontos elemi folyamat (funkció) leírásának.

Az on-line funkciók esetében az inputok és outputok *dialógus* alakjában valósulnak meg. Ennek megvan a maga szerkezete (ld. Dialógustervezés) és neve, amelyet - mint kereszthivatkozást - hozzárendelünk a funkcióhoz. Abban az esetben, ha egyes, a funkciót igénybevevő felhasználók eltérő módon használják azt, előfordulhat, hogy egynél több dialógus készül ugyanarra a funkcióra. Ekkor valamennyire hivatkozni kell.

A funkció egyik legfontosabb célja valamilyen felhasználói igény (lekérdező, vagy karbantartó) kielégítése, amit egyébként a rendszer által nyújtott *szolgáltatás*nak is tekinthetünk. Erre a szolgáltatásra vonatkozóan meg kell határozni az annak színvonalával kapcsolatos követelményt, ami sokmindenre kiterjedhet, kezdve a rendelkezésre állás módjától (pl. éjjel-nappal) a válaszadás idejéig. Az ilyen igények megadására van mód a funkciómeghatározó lap utolsó négy rovatában. Itt meg lehet adni, hogy milyen szolgáltatási szintről van szó (pl. válaszidő), mi ennek a szolgáltatásnak a kívánatos értéke (pl. 3mp), ill. mennyi lehet az ettől való eltérés (pl. max. 10mp). Lehetőség van megjegyzés feltüntetésére is, ami - ha van - többnyire a megengedett eltéréssel kapcsolatos magyarázat.

12.4 A funkciómeghatározás menete

Bizonyos különbséget kell tennünk a lekérdező és a karbantartó funkciók között, mert az előbbiek meghatározása elvileg már igen korán elkezdődhetne (1. szakasz), viszont a karbantartó funkciók csak akkor határozhatók meg, ha már elkészültek a tervezett rendszer logikai AFD-i, ezek pedig csak a 3. szakaszban állnak rendelkezésre. Tekintettel arra, hogy sok karbantartáshoz lekérdezésre is

szükség lehet, valamint a funkció teljeskörű meghatározása - mint az előző pontból kiderülhetett - sok más információ rendelkézesre állását is feltételezi, a funkciómeghatározás helye a 3. szakaszban van.

A funkciók felismerése

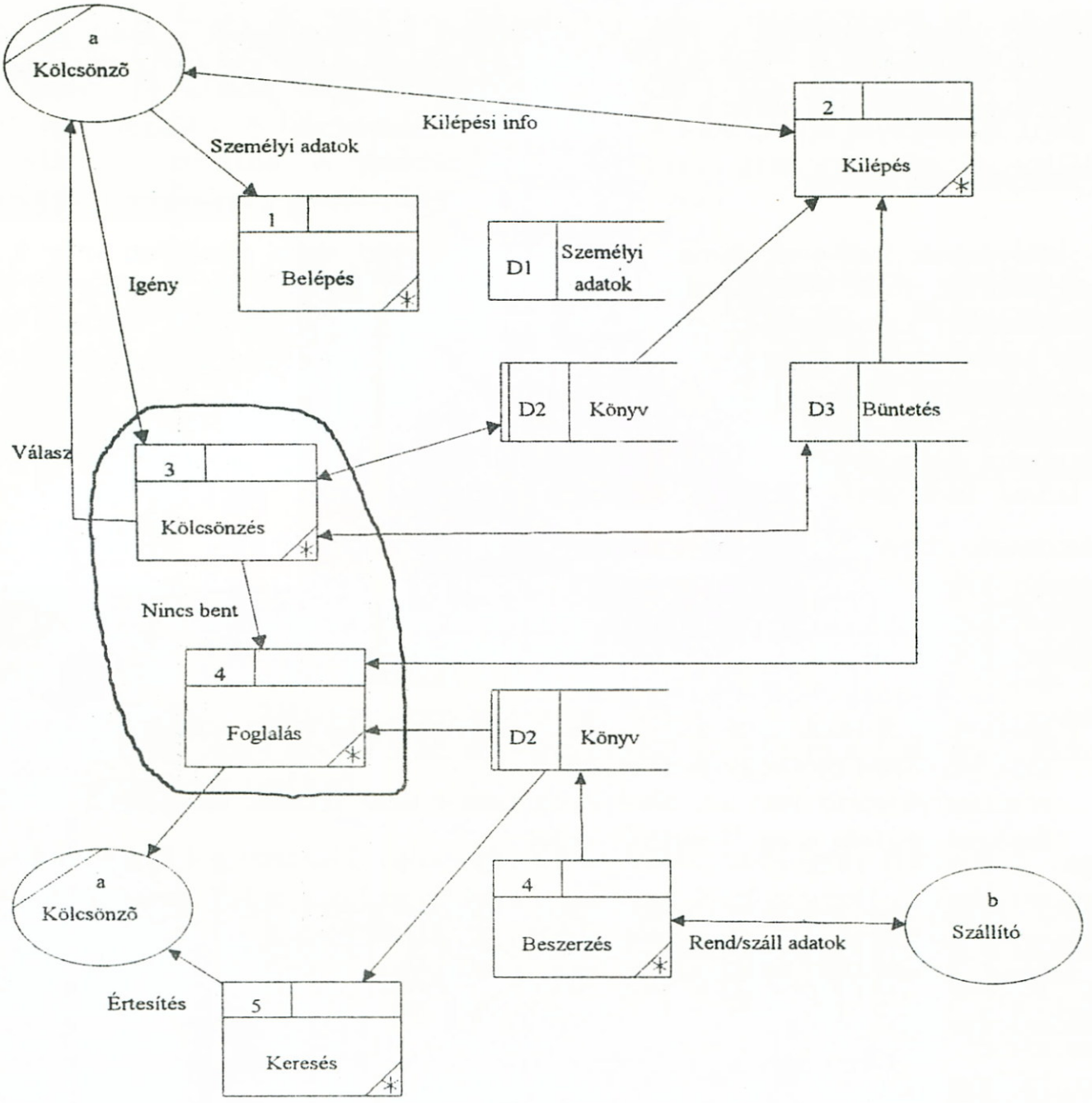
A lekérdezési funkciók felismerése egyszerű, mert ezek általában már korán megjelennek, mint visszakeresési követelmények a követelmény katalógusban. A követelmény katalógusban egyébként természetes módon inkább visszakeresési, mint karbantartási igények jelennek meg. Ez azért van, mert a követelmények a felhasználótól származnak, akinek az *elsődleges* igénye az információs rendszerben tárolt adatok *felhasználására* irányul. Észre kell vennünk, hogy az adatok karbantartása csak szükséges teher, de nem ez az elsődleges cél!

Mindezek alapján teljesen természetes, hogy a karbantartási funkciókat csak később tudjuk felfedezni és szervesen beilleszteni a rendszer működésébe. Ehhez a kiindulópontot a tervezett rendszer legmélyebb szintű logikai AFD-i jelentik. Ezek tanulmányozásakor kétféle funkciót tudunk felismerni:

- *felhasználói funkció* lesz az, amelyet környezeti elemből kiinduló adatfolyam által képviselt esemény indít el, és
- *rendszerfunkció* lesz az, ahol ilyen hatást nem tudunk felfedezni, hanem a funkció végrehajtása "belülről" indul.

A 62. ábrán egy könyvtári kölcsönzési rendszer logikai AFD-jét látjuk. Ha ez szolgál a funkciómeghatározás alapjául, akkor a "Kölcsönzés" felhasználói, míg az "Keresés" (meghatározott időközönként ellenőrzi a kölcsönzési idők lejártát) rendszerfunkció lesz.

A funkciók felismerési folyamatában a két leglényegesebb momentum a kiváltó események végigkövetése, ill. az adatátalakítási szemlélet.



62. ábra

A kiváltó események végigkövetése azt jelenti, hogy mind a felhasználói, mind pedig a rendszerfunkciók esetében végigkövetjük, hogy az esemény (pl. a kölcsönző könyvet kér; eltelik bizonyos idő) "lekezelésének" mi a teljes folyamata, és lehetőleg ezt határozzuk meg funkcióként.

Az adatátalakítási szemlélet azt jelenti, hogy a AFD-n a folyamatokat úgy fogjuk fel, mint a rendszer adatátalakító elemeit, amelyek a bemeneti adataikat vagy környezeti elemektől, vagy az adatmodellből kapják, majd az átalakítás elvégzése után az eredményeket szintén vagy környezeti elemhez, vagy az adatbázisba továbbítják. Az AFD-ken előfordulhat, hogy folyamat-folyamat kapcsolatot is

látunk (ez nincs megtiltva), ami lehet a készítő megértési folyamatának visszatükröződése, vagy az elemi folyamatok egyszerű leírására való törekvés eredménye. A funkciómeghatározásnál arra törekszünk, hogy a folyamat-folyamat kapcsolatokat egy funkcióként értelmezzük. Így a 62. ábra szerinti példában a "Kölcsönzés" és a "Foglalás" nevű folyamatok egy funkcióba kerülnek.

Ez az utóbbi átalakítás, ill. törekvés egyébként összhangban van az adatbázis-szemléletű rendszerszervezéssel, ahol - mint ismeretes - az a cél, hogy minden feldolgozás a közös adatbázisból táplálkozzon, ill. oda helyezze el az eredményeit, ami alól csak a környezeti elemekkel való kapcsolat a kivétel.

A funkciók felismerésének helyességét egyrészt ellenőriztetnünk kell a felhasználóval, másrészt tudnunk kell, hogy ez is egy iteratív, azaz ismétlődően visszatérő tevékenység. Az egyed-esemény modellezés (ld. 14. fejezet) elvégzését követően visszatérünk a funkciómeghatározáshoz, mert ilyenkor pontosabbá válik a képünk a szükséges feldolgozásokról, új funkciókat ismerünk fel és ezt át kell vezetnünk a funkciómeghatározási dokumentáción.

Ugyanígy kell tennünk a prototipizálás során felfedezett funkciókkal.

Végül célszerű végrehajtani egy ellenőrzést arra vonatkozóan, hogy vajon megfelelő-e az események funkciókhoz rendelése. A hozzárendelés szempontjai a következők lehetnek:

- azonos eredet (környezeti elem)
- output generálása azonos környezeti elem számára
- egyidejűség
- azonos egyedtípusokra gyakorolt hatás

Az események funkcióhoz rendelése annál kedvezőbb, minél több pont érvényes rájuk a fenti felsorolásból.

12.5 I/O szerkezetek meghatározása

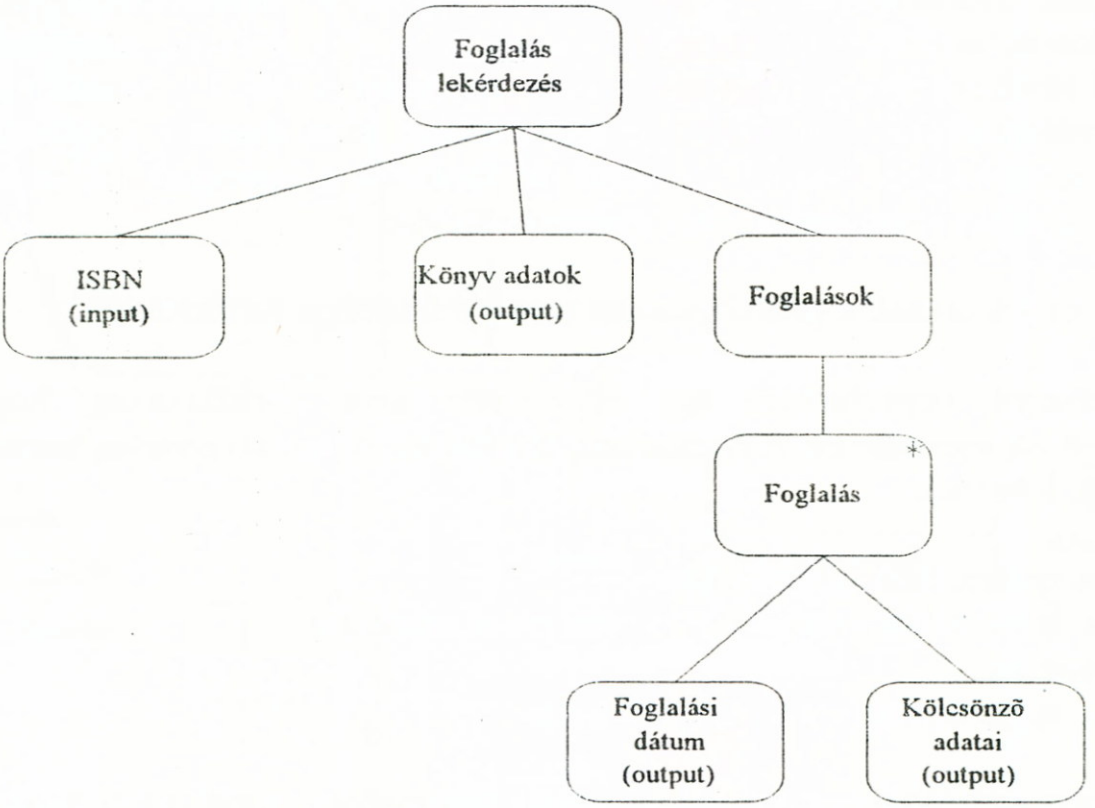
A funkciómeghatározás egyik fontos eleme az inputok és outputok tartalmának meghatározása és az azokban szereplő adattételek logikai szerkezetének feltárása, diagramba foglalása.

Az I/O szerkezeti diagramok készítésénél is az ún. Jackson-féle ábrázolási módot követjük. Erre látunk példát a 63. ábrán, ahol egy visszakeresési funkciónak, a könyvtári könyv-foglalások lekérdezésének I/O szerkezete látható. Értelmezése a

következő: vissza akarjuk keresni egy adott (ISBN-számmal azonosított) könyvre várakozó kölcsönzők adatait, amihez megadjuk az azonosítót (input), mire megkapjuk a könyv leíró adatait - pl. címét, szerzőjét stb. - valamint egy listát a várakozókról. Ez utóbbi ugyanolyan típusú adatok (dátum + kölcsönző adatai) többszöri ismétlődését jelenti, amit jacksoni értelemben iterációként ábrázolunk.

Ha olyan adatok fordulnának elő pl. inputban, amelyeket vagy megadunk, vagy el is hagyhatók, akkor ezek olyan szelekcóként ábrázolhatók, amelyiknek az egyik ága ún. nulla-lehetőséget mutat.

Bizonyos eltérés van az on-line és az off-line funkciók I/O szerkezetének ábrázolásában. Az előbbieknél a párbeszédes jelleg miatt a diagramon váltakozva



63. ábra

jelenhetnek meg input, ill. output adatelemek, míg az utóbbiak esetében ilyen váltakozás nincsen.

Az I/O szerkezeteket azonosító névvel kell ellátni, amire azután hivatkozni lehet a funkciómeghatározás formanyomtatványán.

13. fejezet
A relációs adatelemzés

13.1 A relációs adatelemzés helye és szerepe

Az adatrendszer modellezése SSADM-ben kétféleképpen történik. Amikor az információs rendszer egészéből kiindulva határozzuk meg az egyedtípusokat, mint olyan objektumokat, amelyeket adatokkal (tulajdonságtípusokkal) kell leírni, mert ezek fontosak a felhasználó számára, akkor a logikai adatmodellezés technikáját alkalmazzuk (ld. 10. fejezet). Ezt a technikát - éppen azért, mert mintegy a rendszer egészéből vezeti le a logikai adatszerkezetet - szokás az adatmodellezés felülről lefelé haladó (top-down) módszerének is nevezni.

Ezzel szemben, amikor az információs rendszer fejlesztése eléri azt a szintet, hogy részletes ismeretekkel rendelkezünk a tulajdonságtípusokról, akkor megvizsgáljuk, hogyan függenek ezek egymástól, melyek azok, amelyek meghatároznak más tulajdonságtípusokat, tehát azonosító szerepük van, stb. Ilyen módon mintegy alulról - a tulajdonságtípusok szintjéről - felfelé haladva (bottom-up) jutunk el az egyedtípusok és a közöttük levő kapcsolattípusok meghatározásáig.

A két módszer egymástól független, és mindkettőnek az eredménye egy-egy adatmodell. Tekintettel arra, hogy a két modell ugyanannak az információs rendszernek az adatmodellje, legtöbbször igen nagy mértékben hasonlít egymásra, sőt akár teljesen megegyezik. Amennyiben ez az eset következik be - vagyis megegyeznek - akkor az adatmodellt elfogadhatjuk végleges állapotúnak. Ha eltérés van a két úton kapott eredmény között, akkor alaposan utána kell nézni, hogy mi ennek az oka. Lehetséges, hogy mind a két megoldás jó, de az is lehet, hogy a keresztellenőrzés valamilyen hibát hozott ki. Ekkor a kijavítás után kapjuk a "végleges" megoldást.

Időben mindenképpen a logikai adatmodellezés végezhető el előbb, hiszen ezt - éppen a felülről lefelé haladó jellege miatt - már a fejlesztés korai szakaszaiban alkalmazni tudjuk. A relációs adatelemzés a funkciómeghatározás után kerülhet sorra, mert ekkor már a funkciók inputját és outputját adattétel - azaz tulajdonságtípus - szintjén kell ismernünk. Ez teszi lehetővé a relációs adatelemzés elvégzését.

13.2 A relációs adatelemzés célja

Egyszerűen és röviden fogalmazva: az *optimális adatszerkezet meghatározása.* Ezzel persze egy kicsit meg is kerültük a lényeget, mert hát *mit nevezhetünk optimális adatszerkezetnek?*

Mielőtt erre válaszolnánk egy további pontosításra van szükség: másként kell meghatároznunk az optimalitás fogalmát aszerint, hogy logikai vagy fizikai szintű adatszerkezetet akarunk ebből a szempontból megítélni. A kettő optimuma ugyanis - bár összefügg - nem azonos. A relációs elemzés az adatok logikai összefüggéseivel foglalkozik, következésképpen a logikailag optimális adatszerkezet meghatározásában segít.

Logikailag akkor tartunk egy adatszerkezetet optimálisnak, ha

- valósághű és
- ezt a legtakarékosabban oldja meg.

Az adatmodell nem azért adatmodell, mert az adatokat modellezi, hanem azért, mert a valóságos világ egy részét adatokkal írja le. Ez a feladata. Ahhoz, hogy ezt jól tudja ellátni, nyilvánvalóan pontosan kell visszatükröznie a világ adott részét, ami azt jelenti, hogy azokat az objektumokat kell tartalmaznia a megfelelő tulajdonságaikkal és kapcsolataikkal, amelyekben a felhasználó érdekelt.

A modell "takarékossága" abban nyilvánul meg, hogy csak annyi építőelemet használ fel a fenti cél eléréséhez, amennyi minimálisan szükséges.

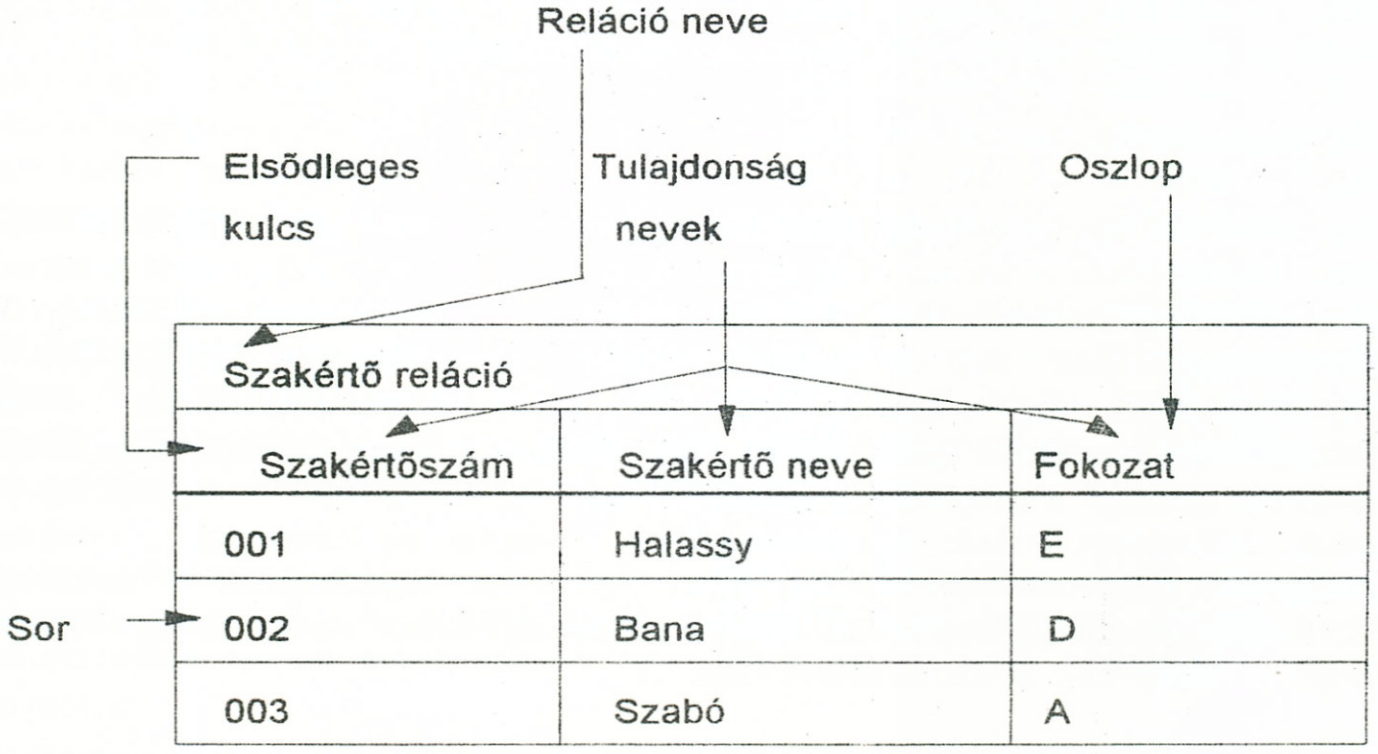
Az adatmodellezésnek a relációs adatelemzésen alapuló irányzata a relációs adatbáziskezelő rendszerek megjelenésével összefüggésben fejlődött ki valamikor a 70-es évek elején. Az alkalmazott fogalmak jelentős része akkorról származik, bár némelyikük egyszerűsödött az évek (évtizedek) folyamán.

13.3 Alapfogalmak

A reláció

A relációs adatelemzés azért "relációs", mert központi fogalma a *reláció.* Eredetileg latin szó, amely a matematikai jelentésén keresztül került át az adatmodellezésbe. A matematikában halmazok elemei közötti kapcsolatot jelent. Codd, a relációs adatbázis kezelés elméletének megteremtője matematikus, és ebbeli minőségében akart "rendet teremteni" az adatszerkezetekkel kapcsolatos "kósza" - sőt valljuk be: olykor kusza - nézetek között. Abból indult ki, hogy

amikor adatrendszert tervezünk, akkor adattételeket hozunk egymással kapcsolatba. Ezt úgy is fel lehet fogni, hogy adatok különféle halmazai között hozunk létre kapcsolatot, vagyis relációt. Ez a kapcsolat technikailag egy kétdimenziós szerkezetben ábrázolható, amelyet - a relációs fogalomkörben - táblázatnak (relációnak) fogunk nevezni. Lássunk erre egy példát (64. ábra).



64. ábra

A kétdimenziós táblázatnak, a relációnak neve van, esetünkben ez SZAKÉRTŐ. A táblázat neve fejezi ki, hogy a benne foglalt adatok a valóságos világ mely objektumát írják le. A táblázatnak az oszlopai - amelyeknek szintén egyedi megnevezésük van - jelentik azokat az adattípusokat (tulajdonság típusokat), amelyeket összefüggésbe hozunk egymással, mégpedig azon az alapon, hogy ezek mind ugyanazt az objektumot jellemzik. Az eredeti értelmezésben ezek voltak az adatoknak azok a halmazai, amelyek között a relációk összefüggést teremtettek (pl. szakértőszámok halmaza, fokozatok halmaza). A táblázat soraiban helyezkednek el a konkrét adatértékek, vagyis a konkrét objektumokra jellemző adatok. Nem nehéz felfedezni a relációban az egyedtípust, tehát azt, hogy *a reláció és az egyedtípus azonos.*

A relációs felfogás értelmében *sem az oszlopok, sem a sorok egymás közötti sorrendjének nincsen jelentősége*. Mivel *nem lehet két azonos sor* egy relációs táblában, fontos, hogy minden egyes sor egyedileg megkülönböztethető legyen. Így tudunk egyértelmű különbséget tenni az egyes konkrét objektumok között. Ehhez szükség van valamire, amivel hivatkozni tudunk a sorokra. Ha van olyan tulajdonságtípus, amelynek értéke minden sorban más és más, akkor a legegyszerűbb, ha ezt használjuk hivatkozás céljára. Úgy is mondhatjuk, hogy ekkor ennek a tulajdonságtípusnak az előfordulásai egyedileg azonosítják az egyes sorokat, tehát ez lesz az objektumok egyedi azonosítója.

Funkcionális függés

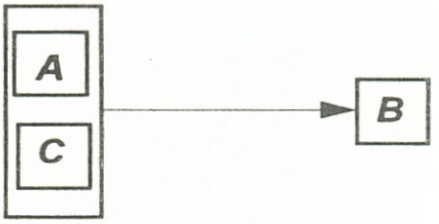
Mit jelent az, hogy egy tulajdonságtípus egyedileg azonosít egy sort a relációban? Első közelítésben a válasz elég egyszerű és megszokott: egy konkrét értékének megadása egyértelműen kijelöli a reláció egy meghatározott sorát, tehát egy sor tulajdonságértéket. Ezt a gondolatmenetet akár meg is fordíthatjuk és kijelenthetjük, hogy amennyiben azt kellene meghatároznunk, milyen tulajdonságtípusokból építsünk fel egy relációt, akkor a legtermészetesebb felépítés azokból a tulajdonságtípusokból adódna, amelyeknek az értékét egy, az adott relációbeli másik tulajdonságtípus - vagyis az azonosító - értékének megadása egyértelműen kijelöli. Más szóval az objektumnak a relációban egymással összefüggésbe hozott tulajdonságtípusai éppen azáltal tartoznak össze, hogy mindegyikük az azonosítótól függ.

Az azonosító is tulajdonságtípus, tehát itt tulajdonságtípusok közötti függésről van szó, ezt nevezzük funkcionális függésnek.

A funkcionális függést - első közelítésben - két tulajdonságtípus között értelmezzük. Legyenek ezek *A* és *B*. Ekkor azt mondhatjuk, hogy *B* funkcionálisan függ *A*-tól, ha *A* egy konkrét értékéhez *B* egyetlen értéke rendelhető hozzá. Ha pl. az *A* a "Személyi szám", *B* pedig a "Személy neve", akkor a leírt függés teljesen egyértelműen felismerhető, valamint az is, hogy ebben a példában a fordított irányú függés nem áll fenn. A gyakorlatban azonban kölcsönös funkcionális függés is előfordulhat.

A funkcionális függés ábrázolása ún. *függésdiagram*on lehetséges:

Ha több tulajdonságtípus (pl. A és C) együtt határoz meg egy másikat (pl. B-t):



Kulcsok

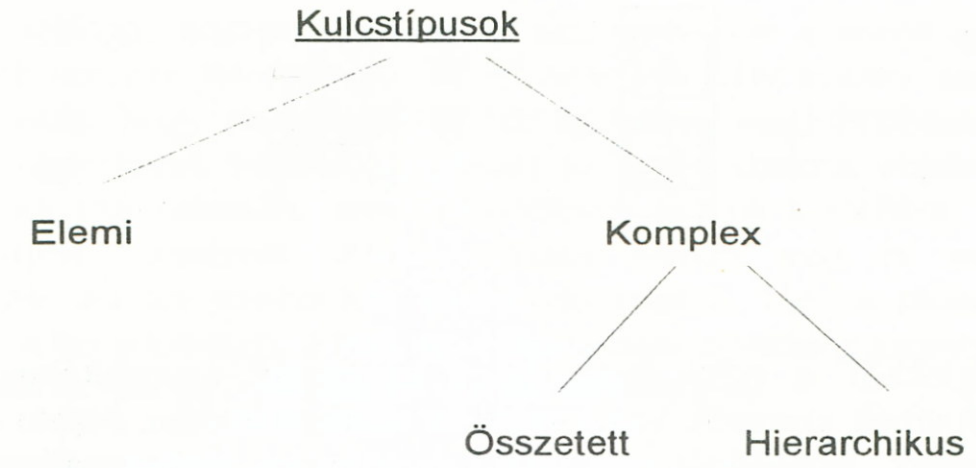
Már az eddigiekből is érzékelhető, hogy az egyedi azonosítóknak - vagyis kulcsoknak - kiemelt szerepük van a relációs adatelemzésben, hiszen ezek azok a tulajdonságtípusok, amelyektől más tulajdonságtípusok funkcionálisan függenek. Ha ezeket megtaláljuk, akkor köréjük csoportosíthatjuk a tőlük függő tulajdonságtípusokat és így eljuthatunk az optimális logikai adatszerkezethez.

Egy relációban *kulcsjelölt*nek tekintünk minden olyan tulajdonságtípust, vagy tulajdonságtípus-csoportot, amely megfelel a sorok egyedi megkülönböztetésének céljára és minimális összetételű. A "minimális" itt azt jelenti, hogy a csoport egyetlen rész-csoportja, vagy része sem képez önmagában kulcsjelöltet. A kulcsjelöltek közül választunk egyet, amely a *kulcs* lesz.

Ha egy relációban előfordul egy másik reláció kulcsjelöltje, akkor ezt *idegen kulcs*nak nevezzük. A reláció saját kulcsjelöltje is kerülhet idegen kulcs szerepkörébe, ha az adott reláció egy másik sorára való hivatkozás céljára használjuk.

Ha egy kulcsjelölt egyetlen tulajdonságtípusból áll, akkor *egyszerű*nek nevezzük. Ha viszont egy vagy több idegen kulcsból tevődik össze, akkor *összetett*nek tekintjük. Az olyan kulcsjelölt, amelynek van egy, vagy több idegen kulcs és egy nem idegen kulcs része, *hierarchikus*.

A kulcs(jelölt)típusok osztályozásának összefoglalását mutatja be a 65. ábra.



65. ábra

A relációk jelölésmódja

A relációkat vagy táblázat alakjában (ld. 64. ábra) vagy a hozzájuk tartozó tulajdonságtípusok olyan jegyzékeként ábrázoljuk, ahol

- a jegyzéknek neve van (a reláció neve),
- egy sorban csak egy tulajdonságtípus szerepel,
- minden, a kulcshoz tartozó tulajdonságtípus alá van húzva,
- az olyan idegen kulcsok előtt, amelyek nem a kulcs részei, csillag áll.

Erre az ábrázolásmódra lehet példa az alábbi:

Megrendelés
Rendelésszám
Rendelés dátuma
Ügyintéző
Szállítási határidő
*Vevőkód

Abban az esetben, ha olyan tulajdonságtípusok is vannak egy relációban, amelyek ott egy előforduláson belül, különböző értékkel többször is ismétlődhetnek - vagyis ún. ismétlődő csoportot alkotnak - akkor, ezeket a tulajdonságtípusokat tabulálással beljebb ugratva tüntetjük fel.

Ha az előző példánál maradva nem csak a rendelés fejléc-adatait akarnánk feltüntetni, hanem ugyanitt a tételeket is, akkor pl. a következő relációhoz juthatnánk:

Megrendelés
Rendelésszám
Rendelés dátuma
Ügyintéző
Szállítási határidő
*Vevőkód
Cikkszám
Cikk neve
Mennyiség
Mennyiségi egység

Az ilyen relációt szokás egymásba ágyazottnak is nevezni, és amint látni fogjuk a modellezés egyik fontos célja, hogy megszabaduljunk tőlük.

13.4 Normalizálás

Az eddigi gondolatmenetből már kiderült, hogy az adatok olyan szerkezetét szeretnénk meghatározni, ahol minden a helyére kerül, vagyis olyan relációink (egyedtípusaink) lesznek, amelyekben *minden nem-kulcs tulajdonságtípus a kulcstól, mégpedig - összetett kulcs esetén - annak egészétől függ funkcionálisan, és más függés nincs a relációban.* Azokat a relációkat, ahol ez teljesül, *normalizált*nak vagy normál alakúnak tekintjük.

Az ilyen adatszerkezet előnye, hogy új objektumok adatainak beillesztése, vagy régiek törlése könnyen és egyértelműen végrehajtható, meglevő objektum tulajdonságtípusainak módosítása pedig csak valamelyik reláció egyetlen sorát érinti (nem kell keresgélni az összes olyan helyet ahol az adott adat előfordul). Ezt úgy is szoktuk mondani, hogy *a normalizált relációkban nincsenek anomáliák.*

A fentiekben meghatározott normál állapot nem automatikusan keletkezik, hanem a tulajdonságtípusok különféle módon előforduló halmazait elemezni kell a funkcionális függések szempontjából, aminek során a meghatározó tulajdonságtípusokból kulcsok lesznek, a meghatározottakat (leíró tulajdonságtípusok) pedig a kulcsok köré csoportosítjuk. Ezekből a csoportokból alakítjuk azután ki az - optimális szerkezetű - egyedtípusokat.

Ebből a gondolatmenetből már valószínűleg érzékelhető, hogy a normalizálás nem egyetlen fejlesztési (elemzési) lépés, hanem egy folyamat. Annak érdekében, hogy ez a folyamat megfelelően átlátható és irányítható legyen, célszerű, bizonyos lépcsőket, ill. ellenőrzési pontokat meghatározni benne. Ezek a normalizálás fokozatai, amelyeket az ún. normál forma fokozattal jellemzünk. Ennek megfelelően létezik *normalizálatlan*, ill. *első, második, harmadik*, sőt magasabb

normál forma. A normál formák hasznos ellenőrző eszközök, amelyek segítenek követni, hogy mennyire közelítettük meg a logikailag optimális (vagyis teljesen normalizált) szerkezetet, de más szerepük nincs. A normalizált állapothoz elvileg a normál formák ismerete nélkül is el lehet jutni.

Normalizálatlan alak

A normalizálás kiindulópontja az I/O szerkezet, amelyet a funkciómeghatározással együtt állítunk elő az SSADM 3. szakaszában, vagyis a követelmények specifikálásakor. Ekkor még többnyire nem ismeretes az inputok, ill. outputok fizikai képe akár képernyők, akár bizonylatok alakjában (ez a specifikáció prototipizálás során történik meg). Olyan I/O szerkezetek állnak tehát a rendelkezésünkre, mint amilyenre a 63. ábra mutatott be példát. Ha az ott látható adatszerkezetet relációként akarjuk leírni, akkor az a következőképpen nézhet ki:

ISBN[9]
Író neve
Könyv címe
Kiadás éve
Kiadó kódja
Kiadó neve
  Kölcsönző száma
  Foglalás dátuma
  Kölcsönző neve
  Kölcsönző címe
  Kölcsönző telefonja

Ez egy olyan reláció, amelyet egyszerűen az I/O szerkezetből írtunk át, anélkül, hogy bármilyen más átalakítást végeztünk volna, vagyis még nem normalizáltunk. Véletlenül persze előfordulhat, hogy egy reláció normalizált, de ezt ekkor még nem feltételezhetjük. Mielőtt azonban elkezdenénk a normalizálást, észrevesszük, hogy egy picit pontosítanunk kell a kiinduló szerkezetet. Egy könyvnek több írója is lehet, tehát itt is ismétlődő csoportot kell feltételeznünk.

Első normál forma

A nem normalizált reláció úgy kerül első normál formába, hogy eltávolítjuk belőle az ismétlődő csoportokat. Az olyan reláció, amelyben nincs ismétlődő csoport, eleve legalább első normál formában van. Az ismétlődő csoportot

[9]International Standard Book Number: nemzetközi szabvány szerinti könyv-azonosító

egyébként azért távolítjuk el, mert az funkcionálisan független a kulcstól, pontosan azéert, mert ugyanazon kulcsérték mellett nem csak egy, hanem több különböző értéket felvevő adat(ok)ról van szó (ld. funkcionális függés meghatározását).

Az ismétlődő csoportot nem egyszerűen "levágjuk" az eredeti relációról, mert akkor elveszne az az információ, ami miatt egyáltalán egy relációba került a nem ismétlődő adatokkal, vagyis a közöttük fennálló logikai kapcsolat. Arról is szó volt, hogy az ismétlődő csoport a relációba ágyazott másik relációként is felfogható és mint ilyennek kell, hogy legyen elsődleges azonosítója, vagyis kulcsa. Példánkban megpróbálhatjuk erre a célra használni a kölcsönző számát, ill. az író kódját, és ekkor a következő három relációhoz jutunk:

ISBN
Könyv címe
Kiadás éve
Kiadó kódja
Kiadó neve

Író kódja
Író neve

Kölcsönző száma
Foglalás dátuma
Kölcsönző neve
Kölcsönző címe
Kölcsönző telefonja

Ebben a megoldásban az író és a könyv, valamint a kölcsönző és a könyv közötti kapcsolat elveszne, amit úgy tarthatunk meg, ha az utóbbi relációknak összetett kulcsot adunk, belehelyezve az ISBN-számot:

ISBN
Kölcsönző száma
Foglalás dátuma
Kölcsönző neve
Kölcsönző címe
Kölcsönző telefonja

ISBN
Író kódja
Író neve

Általános szabályként is kimondhatjuk, hogy az ismétlődő csoportot úgy választjuk le az eredeti relációról, hogy kijelölünk benne egy kulcsot, amelyhez hozzátesszük az eredeti reláció kulcsát, tehát egy összetett kulcsot képezünk számára.

Második normál forma

Azoknak a relációknak az esetében, amelyeknek összetett kulcsuk van, könnyen előfordulhat, hogy valamelyik, bennük szereplő tulajdonságtípus a kulcs egyik részétől önmagában is függ funkcionálisan. Ez nem jó, mert a mi végső célunk - mint fentebb láttuk - az, hogy a leíró tulajdonságtípusok mind a kulcs *egészétől* függjenek.

Az ilyen, ún. részleges függést megvalósító tulajdonságtípusokból ismét új relációt képezünk. Példánkban az íróval kapcsolatos relációban csak egy leíró tulajdonságtípus van, de ez sem az összetett kulcs egészétől függ, és a foglalással kapcsolatos relációban is részleges függést fedezhetünk fel a Kölcsönző száma és a kölcsönző egyéb, személyi jellegű adatai között, ezeket kiemelve új relációkba, a következő eredményhez jutunk:

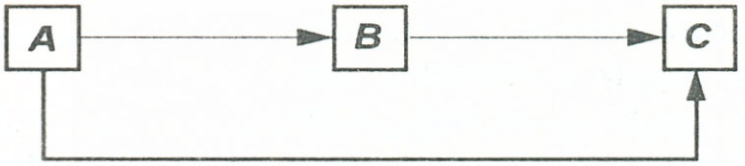
ISBN	ISBN	Kölcsönző száma
Könyv címe	Kölcsönző száma	Kölcsönző neve
Kiadás éve	Foglalás dátuma	Kölcsönző címe
Kiadó kódja		Kölcsönző telefonja
Kiadó neve	ISBN	
	Író kódja	Író kódja
		Író neve

A második normál formához tehát *az szükséges, hogy a reláció már első normál formában legyen, és ne legyen benne részleges funkcionális függés.*

Természetes, hogy minden egyszerű kulccsal azonosított, első normál formában levő reláció egyúttal második normál formában is van, hiszen egyszerű kulcs esetén a részleges függés nem értelmezhető.

Harmadik normál forma

Függetlenül a kulcs felépítésétől, bármelyik relációban előfordulhatnak a leíró tulajdonságtípusok közötti - tehát belsőnek is nevezhető - függések. Ezeket is ki akarjuk küszöbölni, mert egy normalizált relációban *csak* a kulcstól való függés létezhet. Át kell tehát vizsgálni a második normál formában levő relációinkat ebből a szempontból. Ezekben már mindegyik leíró tulajdonságtípus a kulcstól függ, de ezen kívül egymástól való függés is előfordulhat. Függésdiagramban ezt a következőképpen ábrázolhatjuk:



Látható, hogy *B* és *C* egyaránt *A*-tól (a kulcstól) függ, emellett azonban *C* *B*-től is függ. Úgy is fogalmazhatunk, hogy *C* *B*-n keresztül függ *A*-tól, tehát a függése közvetett, idegen szóval tranzitív.

A harmadik normál forma előállításánál az a célunk, hogy a második normál alakú relációkból eltávolítsuk az ott esetleg fellelhető közvetett függéseket. Az ezt megvalósító tulajdonságtípusokból új relációt képezünk.

Példánkban az ISBN-számmal azonosított relációban fedezhető fel közvetett függés a "Kiadó kódja" tulajdonságtípuson keresztül a "Kiadó neve" tulajdonságtípushoz. Ezekből tehát új relációt csinálunk, de természetesen a Kiadó kódja benne marad az eredetiben is, hiszen ez az a tulajdonságtípus, amely a függést közvetíti.

A két új reláció ennek megfelelően a következő alakú:

ISBN	Kiadó kódja
Író neve	Kiadó neve
Könyv címe	
Kiadás éve	
*Kiadó kódja	

Látható, hogy a régi és az új reláció közötti kapcsolatot a Kiadó kódja létesíti, amelyből idegen kulcs lett a régi relációban.

Kimondhatjuk: *a harmadik normál forma feltétele, hogy a reláció már második normál formában legyen és ne legyen benne belső függés.*

Az egész normalizálási folyamatot "nagyüzemi" módon lehet végrehajtani az erre a célra használatos elemző lapokon. Példánkat egy ilyen lapon végigvezetve látjuk a következő oldalon.

A lapon az utolsó oszlop a "Konszolidáció", amelynek szerepéről még szót kell ejteni. A normalizálást az I/O szerkezetekben dokumentált tulajdonságtípusokból kiindulva végezzük el, vagyis I/O szerkezetenként készítünk egy elemző lapot, és ezek mindegyike tartalmaz bizonyos számú harmadik normál formájú relációt. A dolog természetéből következően ezek mindegyikének adattartalma ugyanabból az

információs rendszerből származik, tehát törvényszerűen átfedések lesznek benne. A gyakorlatban azt fogjuk tapasztalni, hogy azonos kulcsú relációkat találunk különböző elemző lapok harmadik normál formájú relációkat tartalmazó oszlopában. Ezeket összegyűjtjük úgy, hogy valamennyi, azonos kulcstól függő leíró tulajdonság egyetlen egyedtípusba kerüljön. Ezt a tevékenységet nevezzük konszolidációnak, az ilyen nevű oszlopban pedig kipipáljuk a már konszolidált relációkat.

Nem norm.	1. NF	2. NF	3. NF	Kon.
ISBN Író neve Könyv címe Kiadás éve Kiadó kódja Kiadó neve Kölcsönző száma Foglalás dátuma Kölcsönző neve Kölcsönző címe Kölcsönző telefonja	ISBN Könyv címe Kiadás éve Kiadó kódja Kiadó neve ISBN Kölcsönző száma Foglalás dátuma Kölcsönző neve Kölcsönző címe Kölcsönző telefonja ISBN Író kódja Író neve	ISBN Könyv címe Kiadás éve Kiadó kódja Kiadó neve ISBN Kölcsönző száma Foglalás dátuma Kölcsönző száma Kölcsönző neve Kölcsönző címe Kölcsönző telefonja ISBN Író kódja Író kódja Író neve	ISBN Könyv címe Kiadás éve *Kiadó kódja Kiadó kódja Kiadó neve ISBN Kölcsönző száma Foglalás dátuma Kölcsönző száma Kölcsönző neve Kölcsönző címe Kölcsönző telefonja ISBN Író kódja Író kódja Író neve	

Magasabb normál formák

Vannak olyan esetek, amikor még a harmadik normál forma sem biztosítja, hogy a reláció normalizált legyen. Gondoljunk pl. arra, hogy a harmadik normál forma nem zárja ki, hogy egy leíró tulajdonságtípus funkcionálisan meghatározza az összetett kulcs valamelyik részét, vagy hogy egy összetett kulcs elemei között nem kívánatos függések létezzenek, de más speciális függéseket sem zár ki ez a forma. Emiatt születtek meg a magasabb normál formák, a *negyedik*, *ötödik*, valamint a *Boyce-Codd normál forma*. Ezekre azonban meglehetősen ritkán van szükség, ezért az SSADM - amely a tömegszerű fejlesztési technológiával foglalkozik - speciális területnek tekinti a magasabb normál formákat. Ha egy projektben szükségesnek látszik foglalkozni velük, akkor erre a feladatra adatmodellezési specialista bevonása szükséges.

13.5 Relációkból diagram

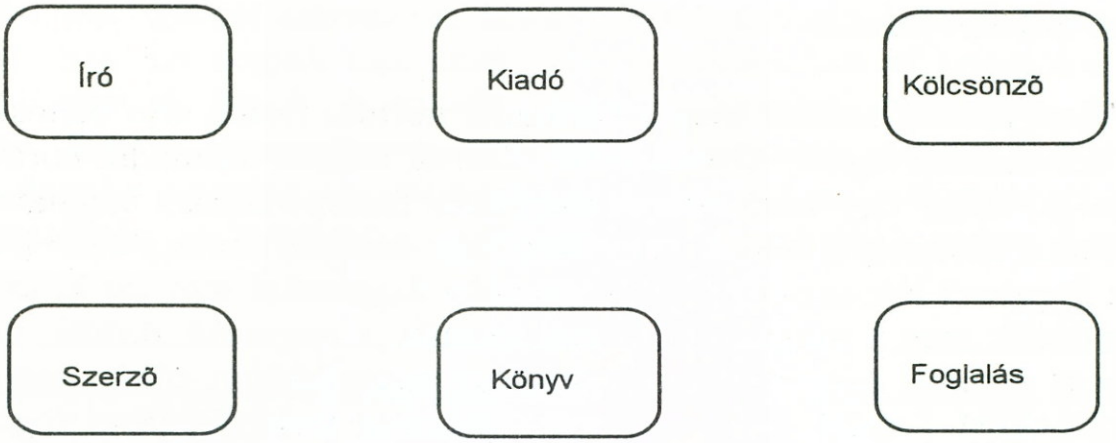
A konszolidálás után a relációs elemzés eredményeként egy harmadik normál formájú relációkból álló relációhalmaz áll rendelkezésünkre. Ebből - néhány egyszerű szabály betartásával - ugyanolyan diagramot rajzolhatunk, mint amilyen a logikai adatmodellezés eredménye. Ezek a szabályok a következők:

1. A relációk egyedtípusok, tehát lekerekített téglalapokkal ábrázolandók.
2. Az idegen kulcsok fölérendelt egyedtípust határoznak meg.
3. Az összetett kulcsok összetevői idegen kulcsok.
4. A hierarchikus kulcs legalább egy összetevője idegen kulcs.

Ezeknek a szabályoknak az alkalmazásával megrajzolhatjuk a példaként normalizált relációinknak megfelelő modell diagramot. Az első szabályt alkalmazva, relációinkat egyedtípusoknak tekintjük, ezért nevet is kell adnunk nekik:

KÖNYV
KÖLCSÖNZŐ
ÍRÓ
SZERZŐ
FOGLALÁS
KIADÓ

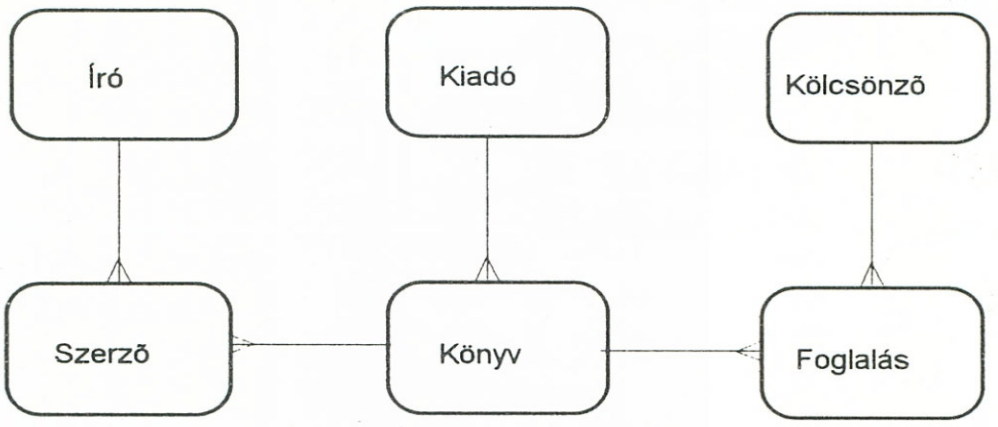
Szerzőnek neveztük el azt a relációt, amely mindössze két kódból áll.
Rajzoljuk fel az egyedtípusokat (66. ábra)



66. ábra

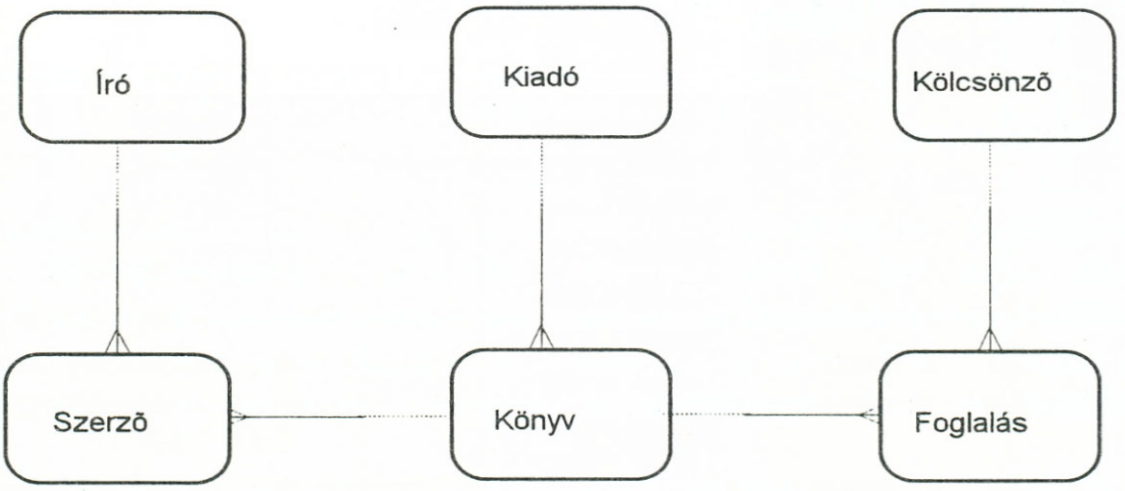
A 2. szabályt alkalmazva felismerhetjük, hogy a KÖNYV fölérendeltje a KIADÓ. A 3. szabály szerint a SZERZŐ-ben és a FOGLALÁS-ban két idegen kulcs van az összetett kulcsban, ezekre ismét alkalmazva a 2. szabályt, azt kapjuk, hogy a FOGLALÁS két fölérendeltje a KÖNYV és a KÖLCSÖNZŐ, a SZERZŐ-é pedig az ÍRÓ és a KÖNYV. Ezeket a kapcsolatokat berajzolva a 67. ábra szerinti diagramhoz jutunk.

Ezen a diagramon még nem látható a kapcsolatok jellege. Ahhoz, hogy ezt is a tulajdonságtípusokból vezethessük le, tudnunk kell, hogy az egyes egyedtípusokhoz tartozó tulajdonságtípusok aszerint is csoportosíthatók, hogy az egyedelőfordulásokban kötelező-e értéket felvenniük vagy nem. Az bizonyos, hogy a kulcshoz tartozó tulajdonságtípusoknak minden egyedelőfordulásban kötelező értékkel rendelkezniük, különben nem lehetnének kulcsok. A FOGLALÁS összetett kulcsára gondolva ez annyit jelent, hogy ennek az egyedtípusnak a szempontjából mindkét fölérendeltje kötelező.



67. ábra

Ha a kiadó kódja kötelező tulajdonságtípus a KÖNYV-ben, akkor a KÖNYV felől ez a kapcsolat is kötelező lesz. Ezekkel a feltételezésekkel a diagram a 68. ábra szerint alakul.



68. ábra

Azt, hogy a kapcsolatok jellege a fölérendelt szempontjából milyen, csak segéd tulajdonságtípusok bevezetésével lehetne tulajdonságtípusokra visszavezetni, amivel e könyv keretei között nem foglalkozunk.

A relációkból akkor tudunk olyan adatmodell diagramot rajzolni, amely összehasonlítható a logikai adatmodellezésnél kapottal, ha valamennyi, de legalábbis kellően sok I/O szerkezetet elemzünk.

A két diagram összehasonlítása során elemezni kell az esetleges eltérések okát és dönteni kell az adatmodell végleges szerkezetéről. Ehhez a most már tulajdonságtípusokkal is teljeskörűen felszerelt modell elérési útjait mégegyszer ellenőrizni kell, mert valójában most dől el egyértelműen, hogy az elérési úton minden kívánt tulajdonság megtalálható-e.

14. fejezet
Egyed-esemény modellezés

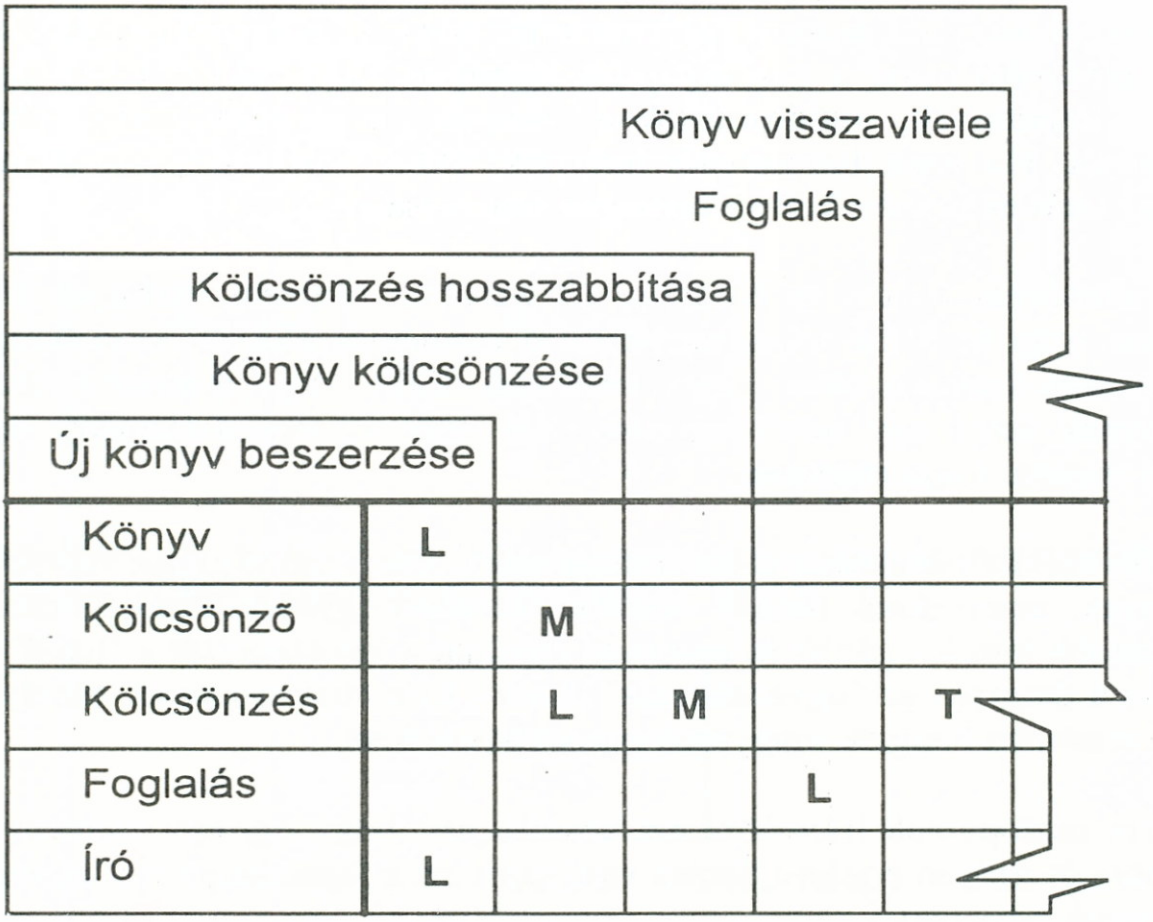
14.1 Bevezetés

Ez a modellezés két technikát tartalmaz, az egyedtörténeti és az eseményhatás diagramok készítését. Ezek alapvető célja az SSADM három dimenziós szemléletében az idő dimenziójának kezelése. Abból indul ki, hogy az információs rendszer időbeli viselkedését a változások határozzák meg, ami teljes összhangban áll az idődimenzió fizikai szemléletével: *az idő a változások révén létezik.*

A változásokat a rendszerben *események* váltják ki, amelyek karbantartó folyamatot indítanak el. Pl. ha új könyvek érkeznek egy könyvtárba, ez egy olyan esemény, amely kiváltja a könyvállományt leíró adatok karbantartását. Az esemény megváltoztatja bizonyos egyed vagy egyedek tulajdonságait (létrehozza, módosítja, vagy törli), ezt nevezzük az esemény által kiváltott *hatás*nak.

Az eseményeket először a tervezett rendszer AFD-in lehet felismerni, mégpedig onnan, hogy vannak adattár felé mutató adatfolyamok a diagramon. Minden ilyen adatfolyam adatkarbantartásra utal, ami pedig változást okoz egy, vagy több egyedben. Az adatfolyam a diagram valamely folyamatából indul ki, az esemény azonban nem azonos ezzel a folyamattal, hiszen a folyamat nem csak egy, hanem több egyed változását is kiválthatja, következésképpen egy folyamat több eseményt is kezelhet. Az eseményeket először a funkciómeghatározás során gyűjtjük össze és rendeljük karbantartó funkciókhoz.

Az események és egyedtípusok közötti kapcsolatot az ún. egyed-esemény mátrixban lehet ábrázolni. Erre mutat példát a 69. ábra.



69. ábra.

A mátrix bejegyzéseinek értelmezése a következő:

L - létrehozás
M - módosítás
T - törlés

Azt mondhatjuk tehát, hogy pl. a Beszerzés, mint esemény létrehozza a KÖNYV egyedet, a Selejtezés pedig törli.

Ez a mátrix igen hasznos szerepet tölt be a rendszerfejlesztési munkában, mert amellett, hogy a bevezetés elején említett két diagramtípus készítésének kiindulópontja, módot ad a karbantartó eseményhalmaz teljességének ellenőrzésére is. Soronként - tehát egyedtípusonként - ellenőrizhetjük ugyanis, hogy nem hiányzik-e valami az adott egyed "életéből". A legdurvább hiba az, ha egy egyedtípus sorában nincs bejegyzés, hiszen az nem valószínű, hogy az adott egyedtípus előfordulásait soha nem kell létrehozni, sem pedig törölni. Hogy módosítani nem kell, az esetleg előfordulhat.

A kedves olvasó már bizonyára felismerte, hogy az a megnyugtató, ha minden sorban előfordul mind a három típusú bejegyzés, mert akkor teljes az egyedek "élete": létrejönnek, módosulásokon mennek át, végül pedig eltűnnek a rendszerből. Ha ezek bármelyike hiányzik, akkor mindenképpen elemezni kell ennek az okát és pótolni kell az esetleges hiányosságot.

Ez az ellenőrzés fontos és hasznos, azt azonban mégsem várhatjuk tőle, hogy elvégzése után biztosan teljes karbantartó eseményhalmazzal rendelkezzünk. Az ugyanis nem tudható előre, hogy létrehozó, törlő, vagy módosító hatása hány eseménynek lehet egy adott egyedre. Ezt csak az egyed-esemény modellezés további menete fogja eldönteni.

A mátrix soraiból készítjük el azt a diagramot, amely azt mutatja, melyek azok az események, amelyeknek hatása van/lehet az adott egyedre. Ezek az események - mint fentebb láttuk - jelölik ki az egyed életútját a keletkezéstől a "halál"-ig. Ezért kapta ez a diagram az "egyedéletrajz" (Entity Life History) nevet, amely azonban szerencsésebben hangzik magyarul *egyedtörténet*ként.

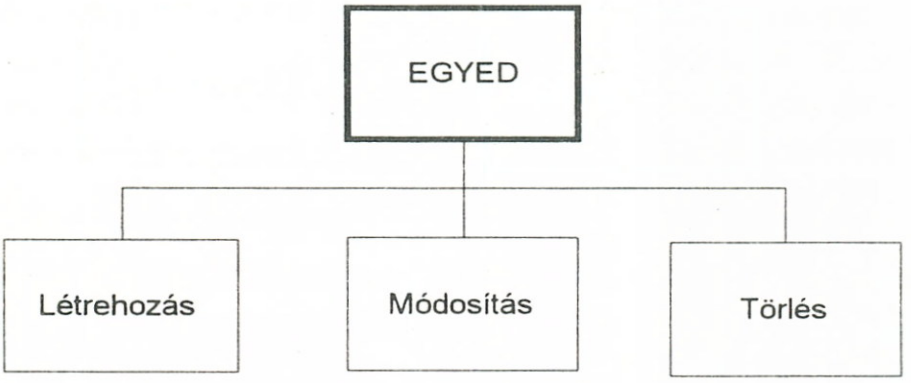
A mátrix oszlopainak irányából azt követhetjük, hogy egy-egy eseménynek mely egyedekre és milyen értelmű hatása van, ezért ezt a fajta diagramot esemény-hatás diagramnak nevezzük.

14.2 Egyedtörténeti diagram (ETD)

Ez a diagram hierarchikus szerkezetű, menden egyes egyedtípushoz egy készül belőle, ezért a hierarchia csúcsán az egyedet jelöljük éles sarkú téglalap alakjában. A hierarchia többi elemét szintén éles sarkú téglalappal ábrázoljuk, és ezek eseményt jelölnek, ha a hierarchia tovább már nem ágazik el belőlük, vagy csoportosító szerepük van a hierarchia alattuk elhelyezkedő része szempontjából.

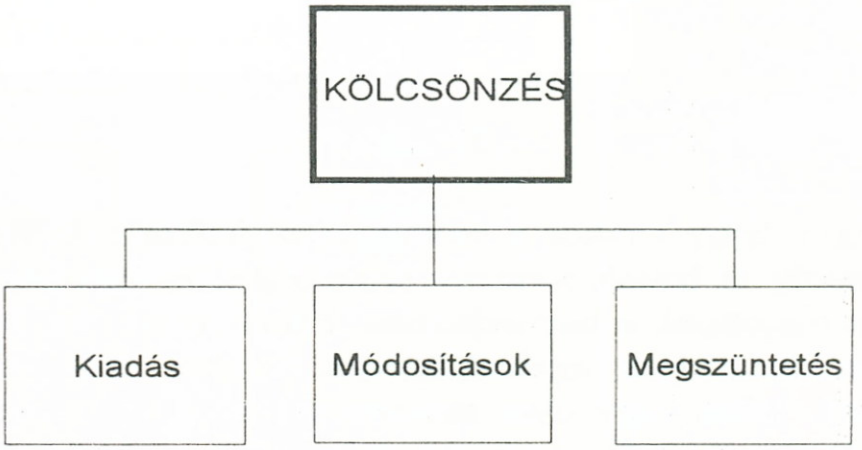
Első közelítésban mindegyik egyedtörténeti diagram (ETD) a 70. ábrán látható módon fogható fel. Ez egyúttal jelzi a diagram olvasásának fő irányát is, ami tehát *balról jobbra*.

Magát a rajzolást is kezdhetjük azzal, hogy a vizsgált egyedtípust jelképező téglalap alá felrajzoljuk ezt a három dobozt, majd sorban megvizsgáljuk, ezek mindegyikét a mátrix alapján, tehát azt, hogy milyen konkrét események sorolhatók az egyes csoportokba a három közül.



70. ábra

Vegyük például a KÖLCSÖNZÉS egyedtípust a könyvtári modellből, és készítsük el ezt az első közelítésű diagramját (71. ábra).

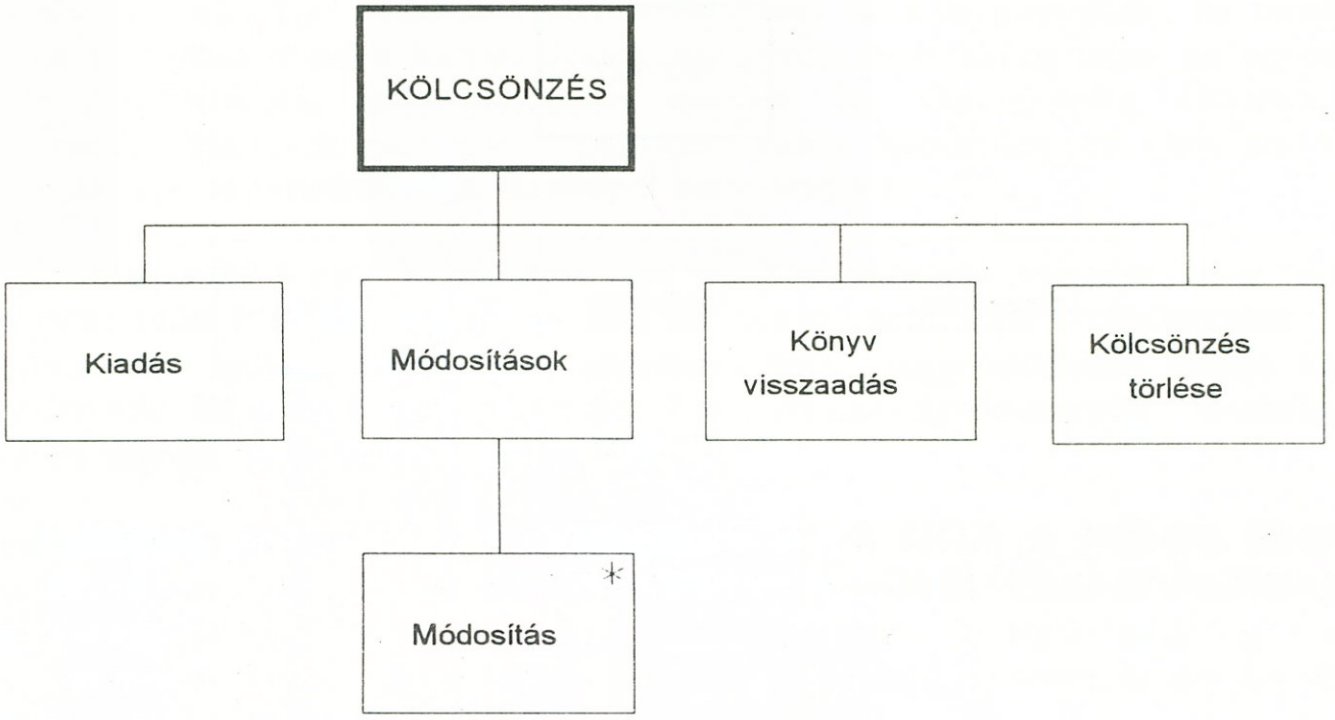


71. ábra

Minden KÖLCSÖNZÉS egyed akkor születik meg, amikor maga a kölcsönzési művelet bekövetkezik, amelynek keretében kiadják a könyvet a kolcsönző személynek.

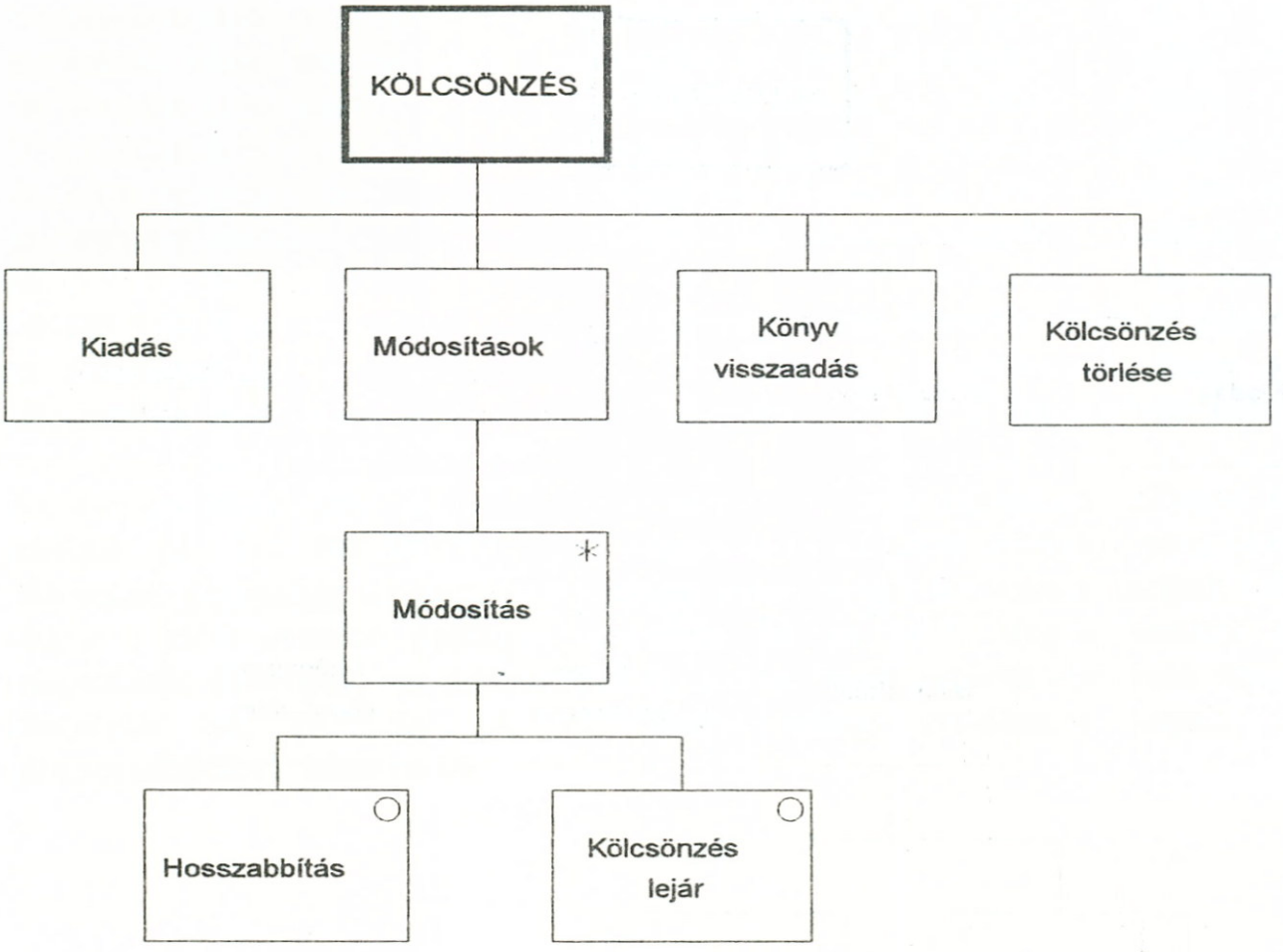
Milyen módosítás lehetséges a kölcsönzéssel kapcsolatban? A legvalószínűbb a kölcsönzés esetleges meghosszabbítása, amire akár többször is sor kerülhet. Ezt a módosítások iterációjaként fejezhetjük ki, tehát ennek megfelelő jacksoni struktúra-elemet iktatunk be (72. ábra).

A KÖLCSÖNZÉS előfordulás megszüntetésével kapcsolatban észrevesszük, hogy előtte van egy lényeges esemény, a könyv visszaadása, a megszüntetés pedig a kölcsönzés törlését jelenti.



72. ábra

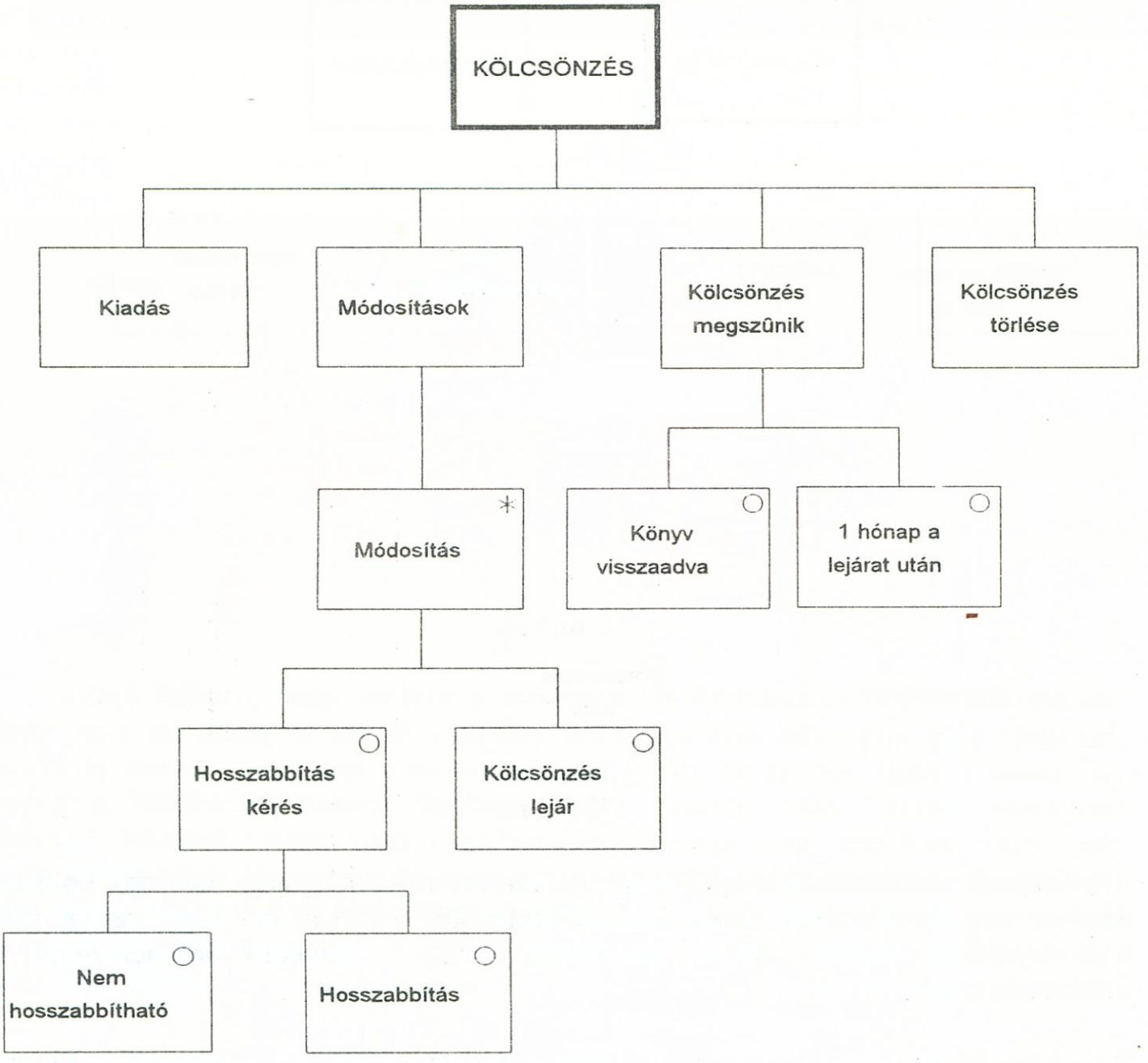
Az ábrából látható, hogy amikor a módosításról felfedeztük, hogy ismétlődnek, akkor nem az addig is létező esemény szimbólumban helyeztük el a csillagot, hanem új szintet nyitottunk a hierarchiában. Ennek az az oka, hogy a szabályok szerint a hierarchia azonos ágában azonos szinten csak "tiszta" szerkezeti elemeink lehetnek, tehát vagy csak szekvencia vagy csak szelekció vagy csak iteráció. Ha tehát észrevesszük, hogy egy már létező szerkezet valamelyik elemét más típusú elemmé kell átalakítanunk, akkor az addigi elem alatt új szintet kell nyitnunk. Ez egyúttal azt is jelenti, hogy ami eddig esemény volt, abból a hierarchia egy belső *csomópont*ja lesz és új esemény vagy események keletkeznek.



73. ábra

A példában a módosítások ágát tovább elemezve rájövünk, hogy egy-egy konkrét esetben vagy az történik, hogy hosszabbítást kér a kölcsönző vagy egyszerűen lejár a kölcsönzési idő. Emiatt ismét - új szinten - szelekciót kell jeleznünk a módosítás alatt (73. ábra).

Az esetet hasonló módon tovább elemezve jutunk ahhoz a véglegesnek tekintett megoldáshoz, amelyet a 74. ábra mutat be.



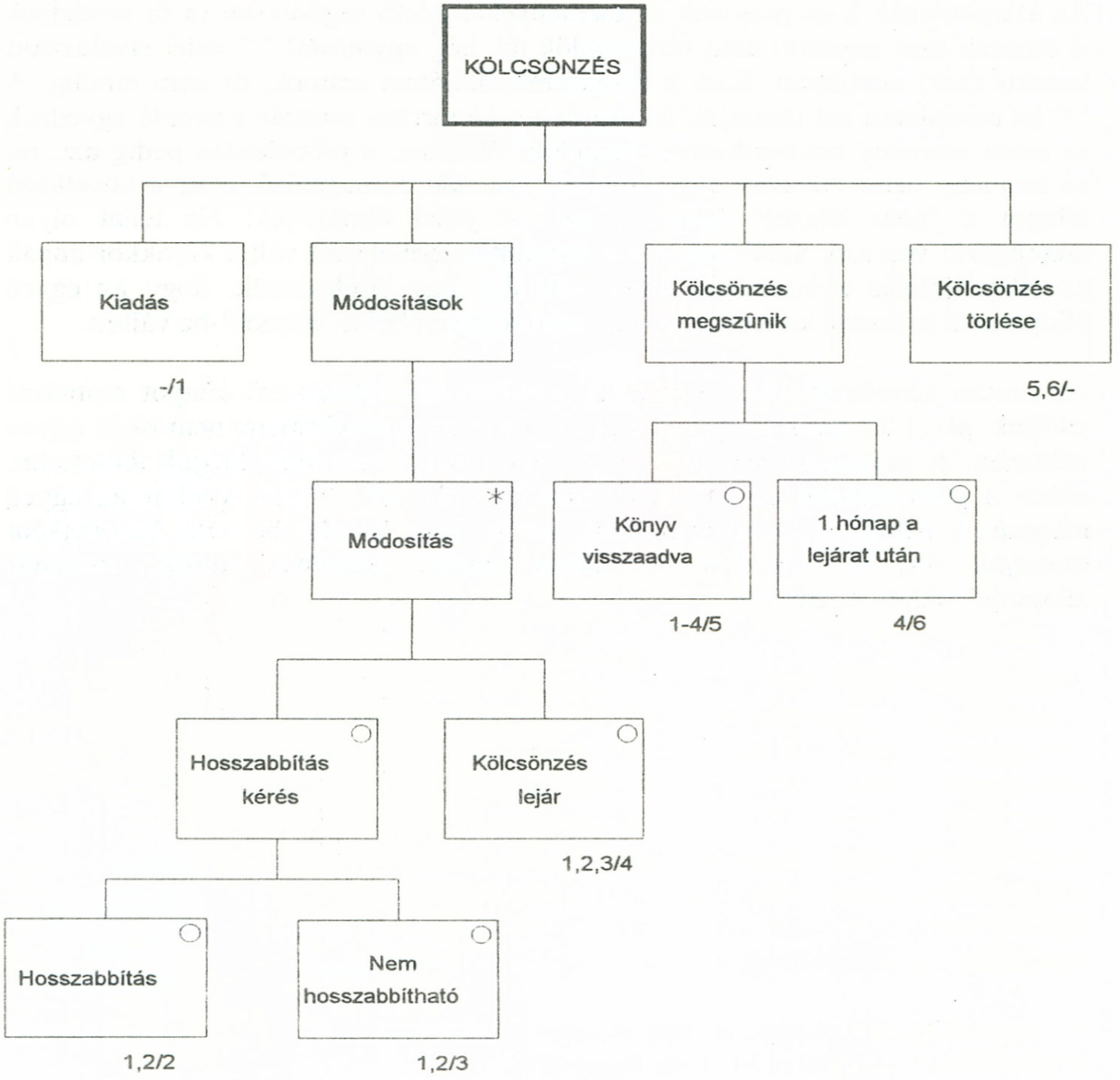
74. ábra

Az egyszerűség kedvéért ez az ETD nem tartalmazza a lejárt kölcsönzés miatti büntetésekkel kapcsolatos eseményeket.

Ennek a fejezetnek a bevezető részében szó volt arról, hogy az események hatása az egyed állapotának megváltozásában tapasztalható, meg arról is, hogy az idő a változásokban nyilvánul meg. Diagramunkban eddig az eseményeket ábrázoltuk, de nem jutott kifejezésre az általuk okozott változás. Ennek követését célozzák az ún. *állapotjelzők.*

Az állapotjelzők a diagramnak az eseményeket jelölő végdobozai (a fa leveleinek is szokták őket nevezni) alatt tüntetendők fel, két, egymástól "/" jellel elválasztott karakter(sor) alakjában. Ezek a karakterek általában számok, de nem mindig. A "/" jel baloldalán azt tüntetjük fel, mi volt a hierarchia csúcsán szereplő egyednek az adott esemény bekövetkezését megelőző állapota, a jobboldalán pedig azt, mi az esemény bekövetkezése utáni állapot. Ha akár a megelőző, akár a következő állapot a "nem létezés" állapota, azt "-" jellel ábrázoljuk. Ha tehát olyan eseményről van szó, amely az egyed létrejöttét (születését) váltja ki, akkor annak az állapotjelzője a következő lehet: -/1. Ezt úgy értelmezzük, hogy az egyed állapota az esemény következtében a "nem létezés"-ből a "létezés"-be váltott.

Az ezután következő változásokat a megelőző és a következő állapot számával jelöljük, pl.: 1/2. Amikor az egyed állapotában az adott esemény nem okoz ugyan változást, de magát az eseményt valami miatt mégis fontosnak tartjuk feltüntetni, akkor a jobb oldalon csillagot tüntetünk fel, pl.: 4/*. Végül, amikor az egyed megszűnik (töröljük az adatbázisból), azt a "nem létezés"-be való átmenetként tüntetjük fel, pl.: 5/-. A 75. ábrán eddigi példánkat látjuk, mostmár állapotjelzőkkel felszerelve.



75. ábra

Próbáljuk meg értelmezni a diagramot. Egy KÖLCSÖNZÉS egyed akkor keletkezik, amikor kiadják a könyvet a kölcsönzőnek, ezzel tehát 1-es állapotba kerül. Ezután módosítások következhetnek, de ne felejtsük el, hogy az iteráció, lehetőségként a nulla számú iterálást is tartalmazza, amikor tehát egyáltalán nincs módosítás. Ha mégis van, akkor ez két okból következhet be: vagy a kölcsönző

kéri a hosszabbítást vagy egyszerűen lejár a határidő. Hosszabbítás kérése esetén vagy megkapja azt a kölcsönző vagy nem. Ha igen, akkor ez az esemény 1-esből 2-es állapotba viszi az egyedet. A hosszabbítás eseményénél azonban az állapotjelző: 1,2/2. Ez azt jelenti, hogy 2-es - vagyis "hosszabbított" - állapotba az 1-es és 2-es állapotok bármelyikéből el lehet jutni, vagyis több, egymást követő módosítás lehetséges.

Ha a könyvtár nem ad hosszabbítást, ez a "nem hosszabbítható", vagyis a 3-as állapotot jelenti. Ide el lehet jutni akár "helyből", ha a könyv az eleve nem hosszabbítható kategóriába tartozik, vagy egyszerűen csak várnak rá mások. A már korábban hosszabbított állapotból is kerülhet ide az egyed, ha már tovább nem lehet hosszabbítani. Itt viszont 3-asból 3-asba nem lehet kerülni, mert amit egyszer nem hosszabbítottak, azt - az adott kölcsönzésen belül - nem is fogják, a diagram legalábbis ezzel a feltételezéssel készült.

A kölcsönzési idő lejárhat, miközben az egyed bármelyik fent felsorolt állapotban van, ez a magyarázata az 1,2,3/4 állapotjelzőnek.

A kölcsönzés kétféleképpen szűnhet meg: vagy visszaadják a könyvet vagy nem, amely utóbbi esetben egy hónap várakozási idő után a megszűnés automatikus. (Az ezzel kapcsolatos adminisztrációval itt most nem foglalkozunk.)

Végül, bizonyos meghatározott idő után a megszűnt kölcsönzéseket fizikailag is törlik az adatbázisból.

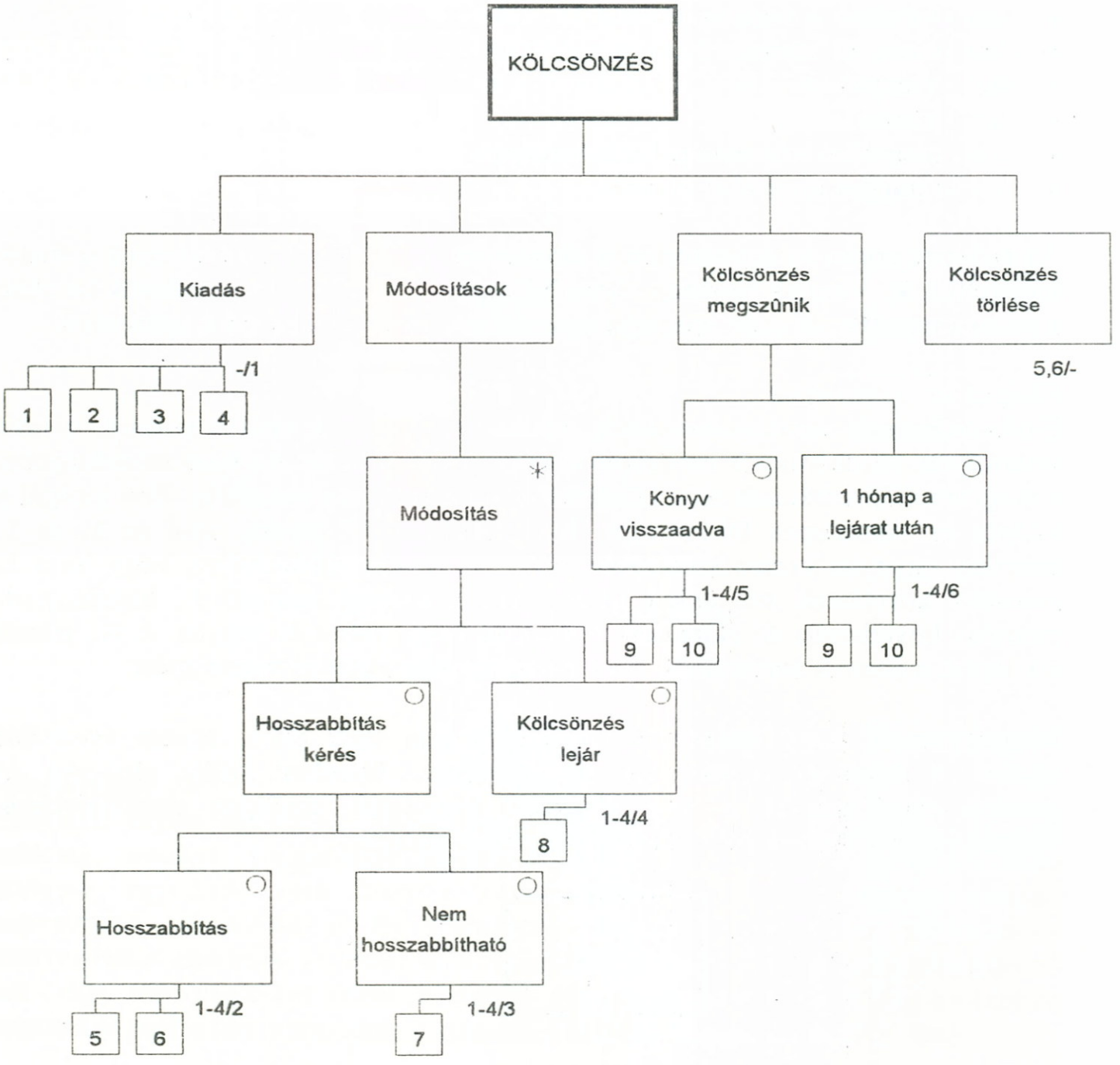
Joggal vetődik fel az olvasóban az a kérdés, hogy mindez érdekes lehet, de mi szükség van rá. Miért jó az, ha ismerjük az állapotjelzőket? Nos, ennek a szerepe negyon hasonló az adatmodellekbeli kapcsolattípusok jellegének (kötelező vagy esetleges) szerepéhez, amelynek feltárásával a későbbi adatbázis tartalmának ellentmondás mentességét (integritását) kívánjuk elősegíteni. Esetünkben ez annyit jelent, hogy egy-egy művelet elvégzése előtt a rendszernek meg kell vizsgálnia az érintett egyed állapotjelzőjének értékét, és csak akkor végezhető el a művelet, ha az megengedett állapot(jelző) változást eredményez. Megengedettek a diagramon feltüntetett átmenetek.

Az események által elindított *műveletek* eredményezik az állapotváltozást. A műveleteket is fel kell tüntetni az egyedtörténeti diagramokon. Ez úgy történik, hogy elkészítjük az adott egyedet befolyásoló események által kiváltott műveletek sorszámozott jegyzékét, majd az egyes események szimbólumaihoz kis négyzetekbe foglaltan csatlakoztatjuk a megfelelő sorszámokat. Példánk diagramját műveletekkel felszerelve mutatja a 76. ábra.

Az diagrammal kapcsolatos műveletek jegyzéke:

1. Kulcsok beállítása
2. Leíró tulajdonságok értékének beállítása
3. Kapcsolás KÖLCSÖNZŐ-höz
4. Kapcsolás PÉLDÁNY-hoz
5. A lejárat dátum helyettesítése aktuális dátum+14-gyel
6. A hosszabbítás jelzőjőnek helyettesítése jelző+1-gyel
7. Foglaltság jelző beállítása
8. Lejártság jelző beállítása
9. Leválasztás KÖLCSÖNZŐ-ről
10. Leválasztás PÉLDÁNY-ról

Vannak megengedett és meg nem engedett műveletek.



76. ábra

A megengedett műveletek körébe tartoznak a már létrehozott egyed tulajdonságai, kulcsai értékének beállítása, későbbi módosítása, valamint más egyedekkel való összekapcsolás, illetve leválasztás más egyedekről.

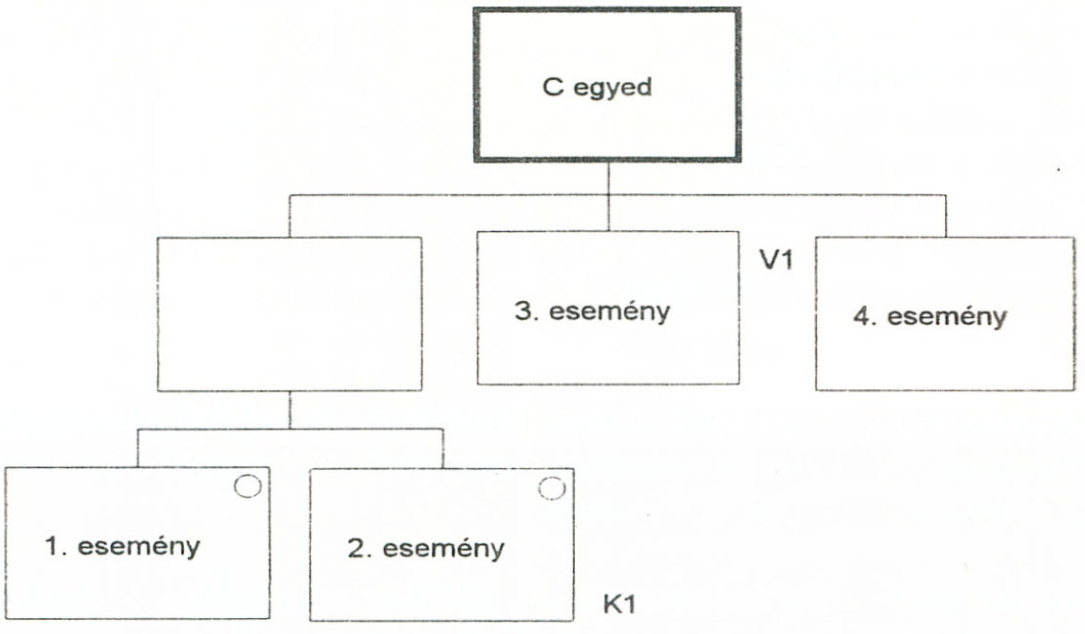
Nem megengedett olyan műveletek feltüntetése az ETD-n, amelyekkel a logikai feldolgozás modellek kialakításakor foglalkozunk. Ezek a következők:

- egyedelérések a bejárási úton
- létrehozás, ill. törlés
- adatérvényesítés
- hibakezelés
- írást megelőző adatmanipuláció/rendezés
- karbantartást megelőző olvasás

Az eddigiek során az egyedtörténeti diagramok készítésének szabályszerű menetét tekintettük át. Vannak azonban olyan helyzetek, amelyek nem sorolhatók be ebbe a munkamenetbe, és ezért speciális megoldásokat, ill. jelölésmódot igényelnek.

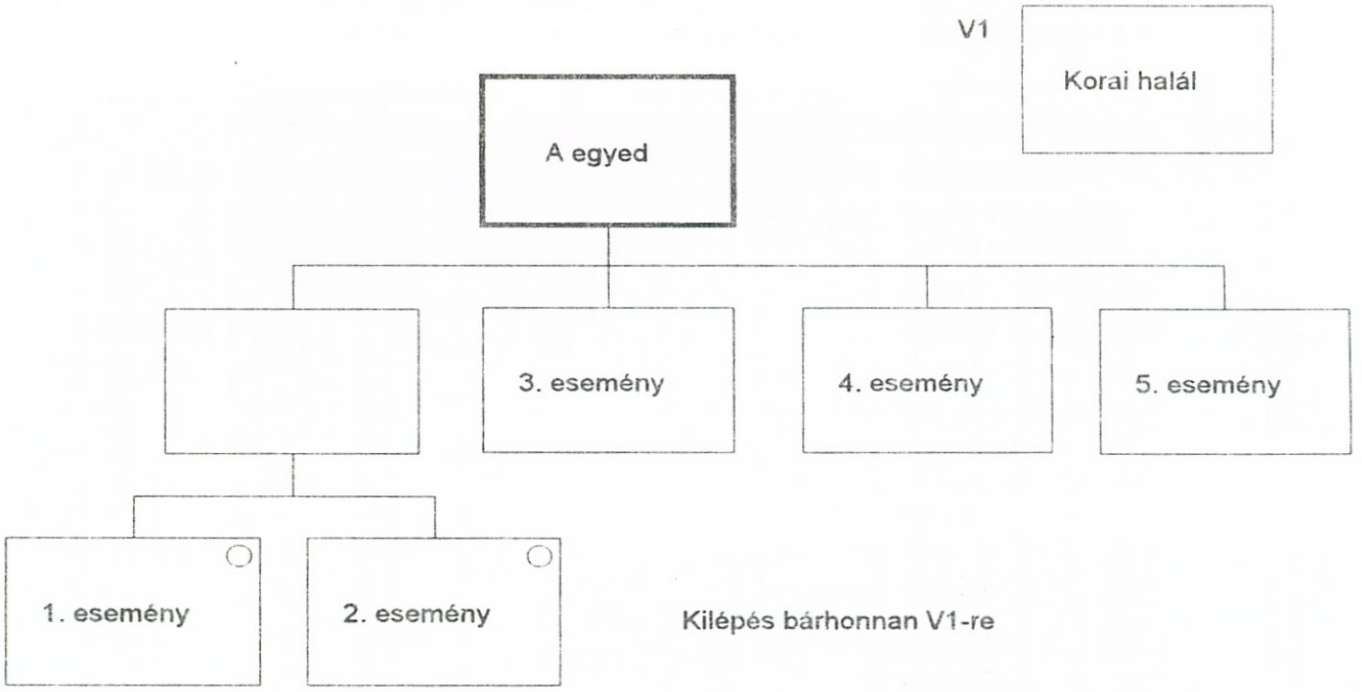
Kilépés és visszalépés
Előfordulhatnak váratlan helyzetek, amikor az egyednek nem a normál, hanem velamilyen, rendkívülinek nevezhető életútja következik be. Az ilyen eshetőségekre a diagram alapszerkezete nem készíthető fel, de azért van megoldás. Erre szolgál a kilépés és visszalépés feltüntetése. Ilyet látunk a 77. ábrán, ahol az 1. és 2. esemény egymást kizáró szerkezetet alkot, és olykor előfordulhat, hogy ha a 2. esemény következik be, akkor utána nem a sorrend szerinti 3., hanem a 4. esemény következik, de előre nem tudhatjuk, hogy mikor és hogyan. A K1 jelenti a kilépést a normál sorrendből, a V1 pedig az ehhez tartozó visszalépést.

Megjegyzendő, hogy kilépés és visszalépés nem csak előre, hanem visszafelé ugrást is jelenthet a diagram szerkezetében. Ilyenkor külön elemzést és gondoskodást igényel a már elvégzett műveletek esetleges eredményeinek visszaállítása a visszalépés helyének megfelelő állapotra.



77. ábra

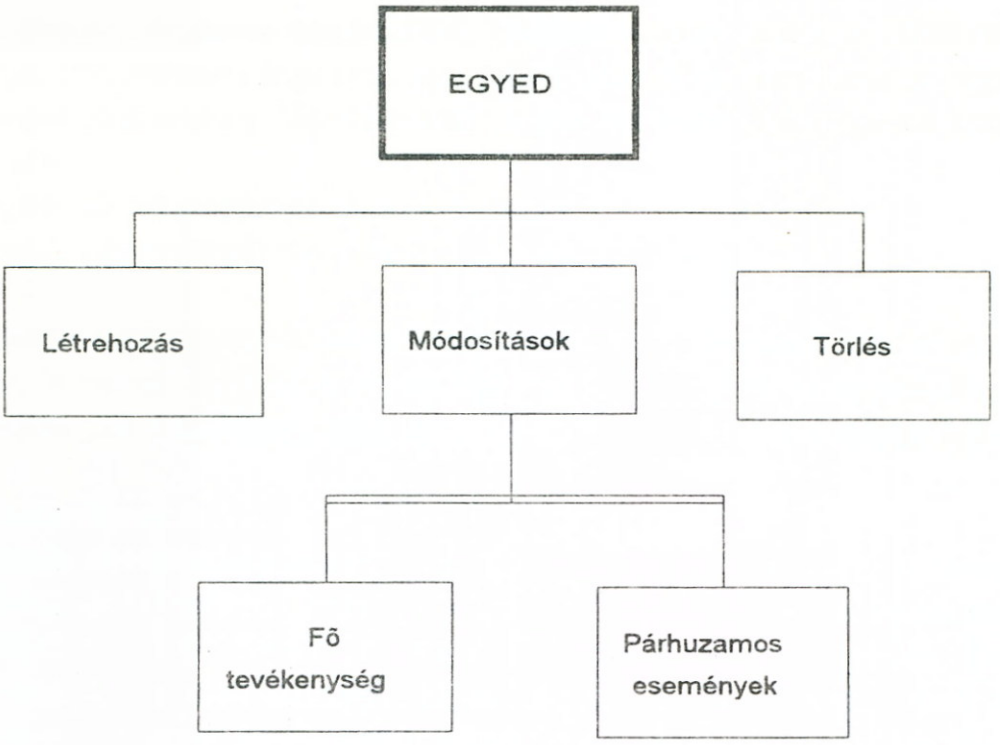
A kilépés további sajátos esete, amikor bárhonnan a szerkezetből bármikor ki lehet lépni egy, a szerkezeten kívüli, általános érvényű eseményre, amely egyúttal meg is szünteti az egyedet. Szokták ezt a "korai halál" esetének is hívni (78. ábra).



78. ábra

Párhuzamos élet

Lehetnek olyan egyedeink, amelyeknek az életében olyan alszerkezetekre van szükség, amelyek függetlenek egymástól. Ilyen lehetne pl. ha a diagramban ábrázolt egyednek nem közvetlenül az ETD által leírt főtevékenységgel kapcsolatos változásait is fel szeretnénk tüntetni. Ilyenkor ún. párhuzamos szerkezeteket alakítunk ki, amelyek a hierarchia egymással valóban "párhuzamos" két (vagy több) ágaként jelennek meg és a jó megkülönböztethetőség érdekében kiindulási pontjaikat párhuzamos vízszintes vonalak fogják össze (79. ábra).



79. ábra

Végül felhívom az olvasó figyelmét arra, hogy az ETD-k szerkezetét befolyásolhatja az adatmodell szerkezete is, mégpedig az egyedtípusok alá-fölérendeltségi viszonyai, ill. a kapcsolatok kötelező vagy esetleges volta. Ha pl. egy kapcsolat az alárendelt felől kötelező, akkor a fölérendelt törlését mindenképpen eseményként kell feltüntetni az alárendelt ETD-jében, mert ez az esemény vagy törli az alárendeltet is vagy egy másik fölérendelt előforduláshoz kell "átkötni" a vizsgált egyedet, ami szintén esemény.

14.3 Eseményhatás diagramok (EHD)

Ezeknek a diagramoknak az elkészítésekor az egyed-esemény mátrix esemény-oldalából indulunk ki és az adott eseménynek az egyedekre gyakorolt hatását követjük a diagrammal. Az események - mint bizonyára emlékszik az olvasó - karbantartó jellegűek, tehát ezek fogják elindítani egy vagy több egyed módosítását, hatásaikat végigkövetve és ezt a diagramtípust elkészítve alapozzuk meg a karbantartó feldolgozások logikai modellezését.

A munka menete a következő lépésekben történik:

- Az egy esemény által érintett egyedtípusok dobozainak felrajzolása
- Szelekció és/vagy iteráció feltüntetése
- Az ún. 1:1 összefüggésben levő hatások jelölése

- Az iteratív hatások összevonása ott, ahol ez lehetséges
- Az eseményt képviselő adat(ok) feltüntetése a diagramon

A továbbiakban röviden áttekintjük ezeket a lépéseket.

Egyedtípusok jelölése
A legegyszerűbb esetben ezek az egyed-esemény mátrixból kiolvashatók és ennek megfelelően felrajzolhatók, mint lekerekített sarkú dobozok. A mátrixból történő kiolvasás és felrajzolás után figyelembe kell venni az egyedtörténeti diagramok megalkotásakor szerzett tapasztalatokat is. Ilyen tapasztalat lehet, hogy egy esemény esetleg nem egy, hanem több előfordulását is befolyásolja egy egyednek, de különbözőképpen. Ha erről van szó, akkor ugyanazt az egyedet többször is fel kell tüntetnünk készülő diagramunkon. Egyéb, a dobozoknak a rajzon való elhelyezésével kapcsolatos előírás nincs.

Szelekció és/vagy iteráció
Ha egy esemény többféleképpen hathat egy egyedre, annak tartalmától (tulajdonságértékeitől) függően, akkor ezeket a hatásokat jacksoni értelemben vett szelekcióként ábrázoljuk a kérdéses egyed doboza alatt.

Hasonlóképpen, ha az esemény következtében az érintett egyedtípus több előfordulását is karban kell tartani, akkor az egyedtípus dobozát felcseréljük egy, az egyedtípust csoportosító dobozzal, amely alatt iterációként tüntetjük fel a többszörös karbantartást.

A hatások megfeleltetése
Ez talán a leglényegesebb művelet a diagram kialakítása során, hiszen ekkor kapcsoljuk össze egymással - vagyis feleltetjük meg egymásnak - azokat az egyedeket, ill. csoportjaikat (iterációnál), amelyekre az adott esemény hatással van. Innen származik a diagramtípus eredeti angol neve: Effect Correspondence Diagram = Hatásmegfeleltetési diagram. Az ilyen módon összefüggésbe hozott diagram-elemekről azt mondjuk, hogy 1:1 megfeleltetésben állnak egymással.

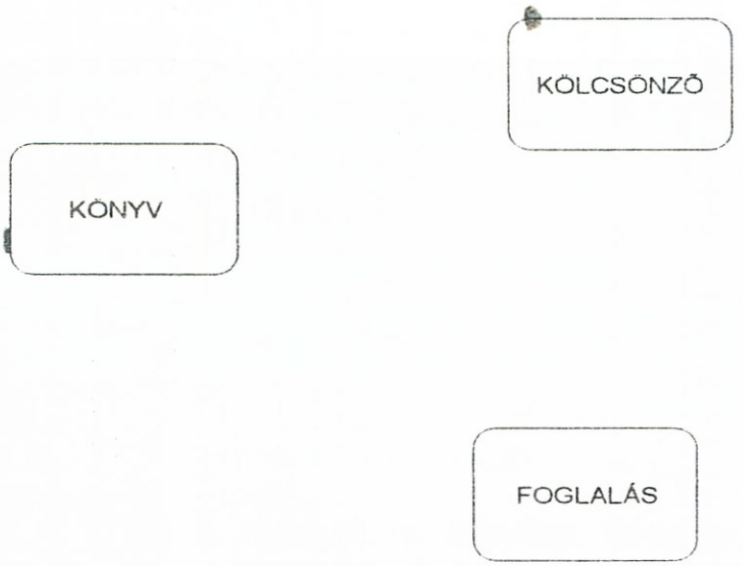
Iteratív hatások összevonása
Ha egy egyedtípus dobozához két, vagy több olyan iteráció kapcsolódik, amelyek ugyanazon a kapcsolattípuson alapulnak, akkor ezeket össze kell vonni egyetlen iterációs dobozba.

Az eseményadatok feltüntetése
Ugyanúgy, ahogy a lekérdezési út modellezésénél fel kellett tüntetni a belépéshez szükséges adatokat, itt is jelezni kell az eseménynek azokat az adatait, amelyek elindítják az érintett egyedek karbantartását. Ebből az is látható, hogy a

lekérdezési út modelljének a karbantartások esetében az esemény-hatás diagram a megfelelője. Úgy is mondhatnánk, hogy az EHD a "karbantartási út" modellje.

Próbáljuk meg egy egyszerű példán követni a fentiekben kifejtett lépéseket! A példa ismét a már ismert könyvtári környezetből származik. *Legyen az esemény a könyv-foglalás, ami a kölcsönző igényére történik azokra a könyvekre, amelyeket meg szeretne kapni, de a kölcsönzési szándék pillanatában nem állnak rendelkezésre.*

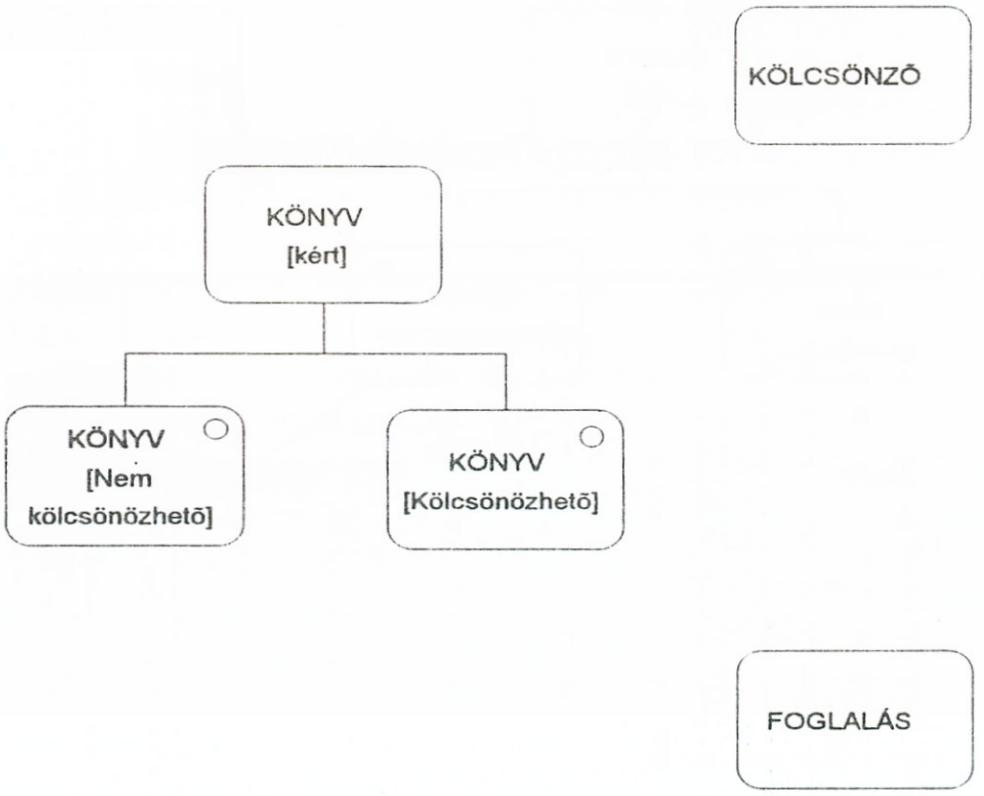
Az adatmodell az 55. ábra szerinti, és ebből igen könnyen kiolvasható - akár az egyed-esemény mátrix nélkül is - hogy melyek az érintett egyedtípusok: KÖLCSÖNZŐ, KÖNYV, FOGLALÁS. Ennek alapján felrajzolhatók az egyedtípusok dobozai a kezdő diagramra (80. ábra).



80. ábra

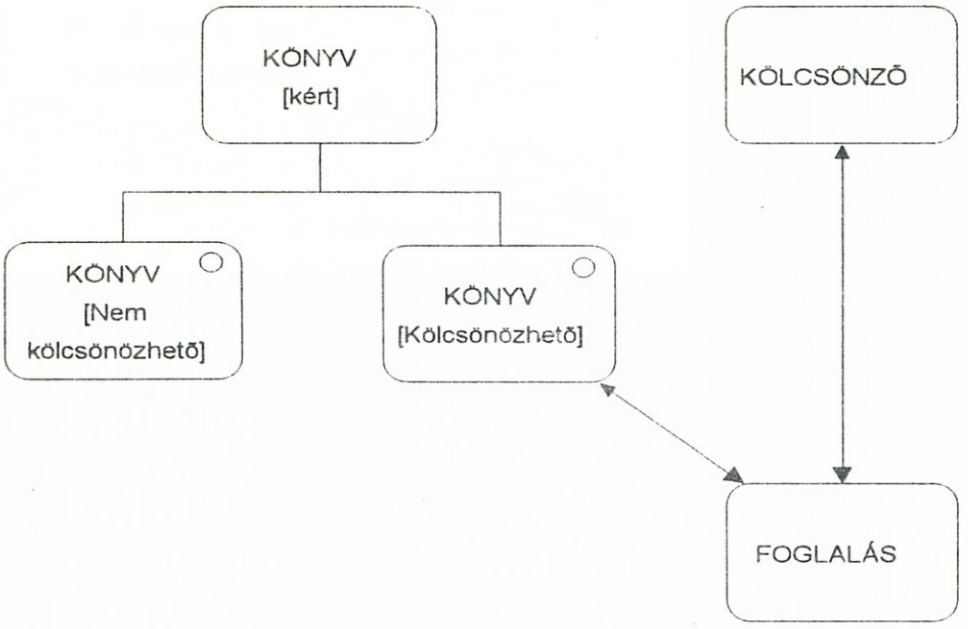
A következő lépésben azt keressük, hogy ez a karbantartás tartalmaz-e iterációt, vagy szelekciót. Mivel a folyamat a kívánt könyv megkeresésével kezdődik, és még ha több könyvet is akarunk foglalni, ez mindig egyenként történik, egy kölcsönző számára, egy foglalásként, ezért itt iteráció nincs.

Lehet viszont szelekció, ha azt feltételezzük, hogy a könyvek kölcsönözhető, ill. nem kölcsönözhető kategóriákba vannak sorolva (81. ábra).



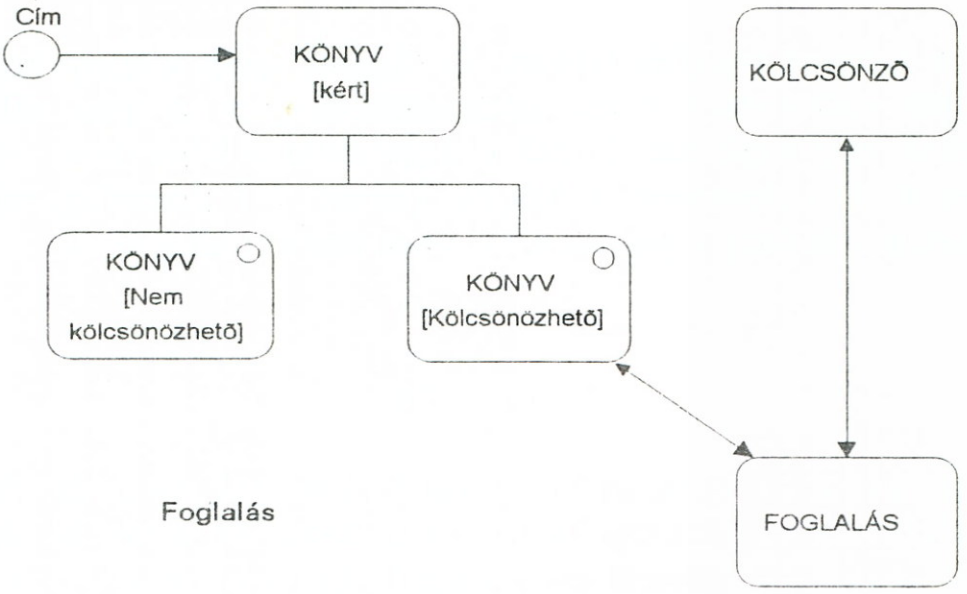
81. ábra

Nézzük most a hatások megfeleltetését! Be fogunk lépni a könyvhöz, mondjuk a kért cím alapján, aztán eldől, hogy kölcsönözhető-e, és ha igen, akkor innen 1:1 alapon készítünk egy foglalást (hozzá is kapcsolva a könyvhöz), majd ezt az 1 foglalást szintén 1 kölcsönzőhöz kötjük (ismét 1:1 összefüggés). Ezeket az összefüggéseket is feltüntetve jutunk a 82. ábra szerinti állapothoz.



82. ábra

Végül tüntessük fel az eseményt jelképező belépési információt és írjuk is fel a diagramra az esemény nevét (83. ábra).



83. ábra

Ezzel eseményhatás diagramunk teljessé vált.

15. fejezet
Logikai feldolgozástervezés

15.1 Bevezetés

Ennek a technikának (amely tulajdonképpen kettő) az eredeti neve Logical Database Process Design = logikai adatbázis feldolgozás tervezése, ami jól tükrözi az SSADM adatközpontú szemléletét. Hasonlít ez az adatfolyam diagramok felfogásmódjára, ahol az adatok források és nyelők között áramlanak, közben pedig itt-ott átalakítják vagy tárolják őket.

Az információs rendszer feldolgozás-oldalát az SSADM elsősorban az adat-oldalhoz viszonyítva közelíti: abból indul ki, hogy *vannak adatok*, és ezeket *vissza lehet keresni*, ahhoz pedig, hogy a visszakeresés során a valóságnak megfelelő információt kapjunk, az adatokat *karban kell tartani*. Az adatbázis-feldolgozás tehát ebben a szemléletben az adatok visszakeresését és módosítását jelenti.

Persze az adatokon lehet olyan feldolgozásokat is végezni, amelyek nem közvetlenül az adatbázissal kapcsolatosak. Ilyenek a tárolást megelőző, vagy a visszakeresést követő számítási vagy logikai műveletek. Ezeket az AFD-k elemi folyamataiként kell meghatározni.

Ennek megfelelően a logikai adatbázis-feldolgozások tervezése két fő részből áll: az adatbázis-lekérdezések, valamint az adatbázis-karbantartások logikai tervezéséből. Az egyszerűség kedvéért a továbbiakban lekérdezések logikai tervezéséről, ill. karbantartások logikai tervezéséről fogunk beszélni.

A feldolgozások logikai tervezésének két célja van:

- a Követelményspecifikáció moduljában összegyűjtött információ olyan megfogalmazása, ami a fizikai tervezés alapjául szolgálhat,
- a rendszer olyan, logikai meghatározása, ami bármikor később segít a szükséges karbantartások végrehajtásában.

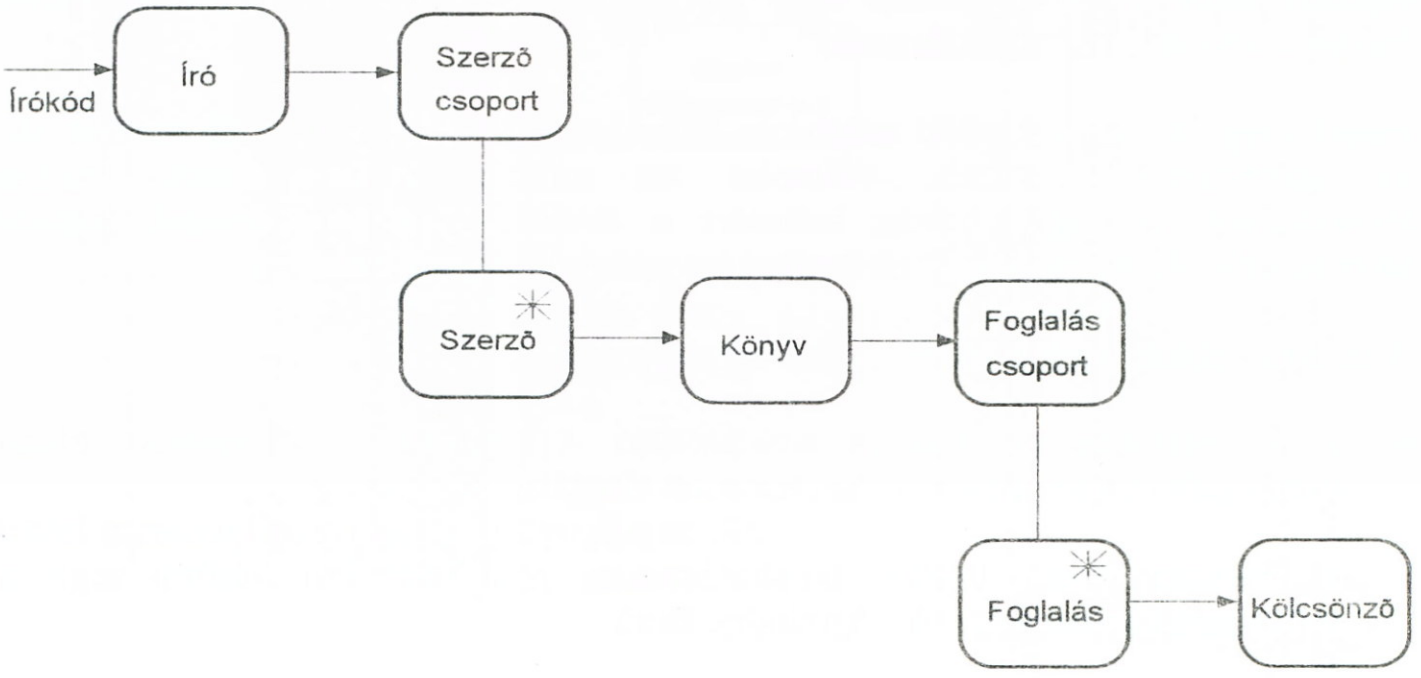
15.2 Lekérdezések logikai tervezése

Ez a technika a lekérdezési funkciókhoz kapcsolódó feldolgozás logikai tervezését jelenti. Abból indul ki, hogy a lekérdezési funkció inputját a bejárandó elérési út specifikációja alkotja, outputja pedig a funkcióra meghatározott I/O szerkezet output oldala. A kettőnek összhangban kell lennie egymással, hiszen a bejárási úton kell "összeszedni" a kívánt output elemeit.

Végeredményben ez képezi ennek a feldolgozásnak a lényegét, tehát ha meg tudjuk feleltetni egymásnak a fentiek szerinti input és output elemeit, akkor megkapjuk az elvégzendő feldolgozás logikai elemeit.

Az egész tervezési folyamatot a legérthetőbben egy példán keresztül lehet bemutatni. Erre a célra ismét a könyvtári környezetet használjuk fel, mégpedig az elérési út modellezésével kapcsolatban ismertetett visszakeresést. Tekintsük tehát át a szükséges tervezési lépéseket ezen a példán, amely tehát úgy szólt, hogy *vissza akarjuk keresni mindazokat a kölcsönzőket, akik egy meghatározott író műveire várakoznak.*

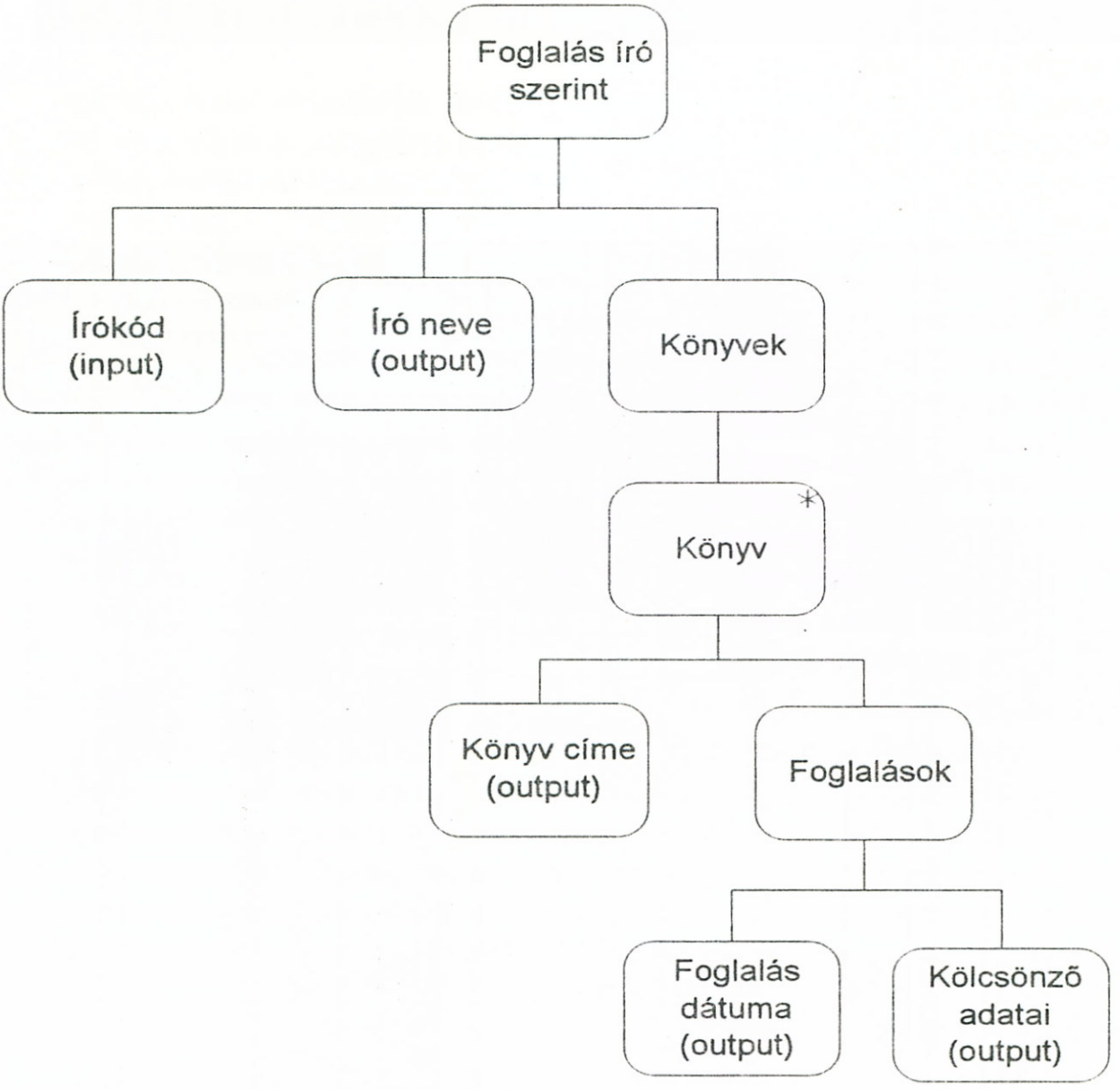
1.) A lekérdezési út meghatározása
A lekérdező feldolgozások modellezésének - mint említettem - a lekérdezés út meghatározása az egyik kiindulópontja. A 11. fejezet tartalmazza a példa lekérdezési útjának meghatározásával kapcsolatos részleteket, ezért itt csupán a végeredményt, vagyis a lekérdezési út modelljét ismételjük meg (84. ábra).



84. ábra

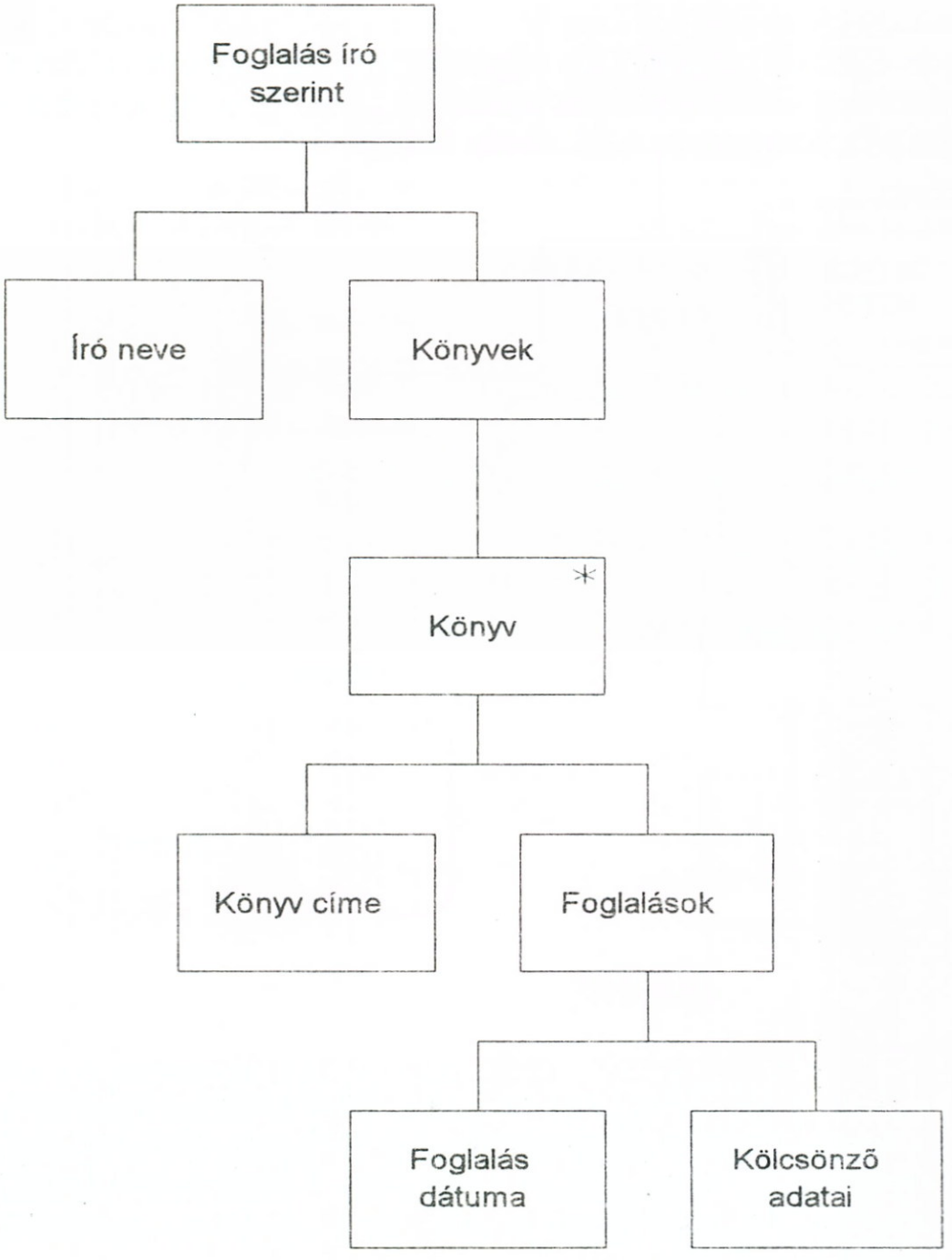
2.) A lekérdezés outputjának meghatározása
Ez lesz a modellezés másik kiindulópontja. Ha a fejlesztési folyamat időbeli menetét tekintjük, akkor megállapítható, hogy mire a logikai tervezéshez érünk, addigra elvileg már készen kell lennie az I/O szerkezetek terveinek az egyes funkció meghatározásokhoz rendelt módon. Emlékezzünk vissza: az I/O szerkezetek a funkciómeghatározásokkal együtt az SSADM 3.számú szakaszában készülnek, míg a feldolgozások logikai tervezése a 4. számú szakasz feladata.

A funkció meghatározással foglalkozó 11. fejezetben szerepel példa az I/O szerkezet meghatározásának bemutatására, és az szintén a könyvtári környezetből származik, de nem egészen ennek a visszakeresésnek az I/O szerkezete. Az itt szükséges szerkezetet a 85. ábra mutatja be.



85. ábra

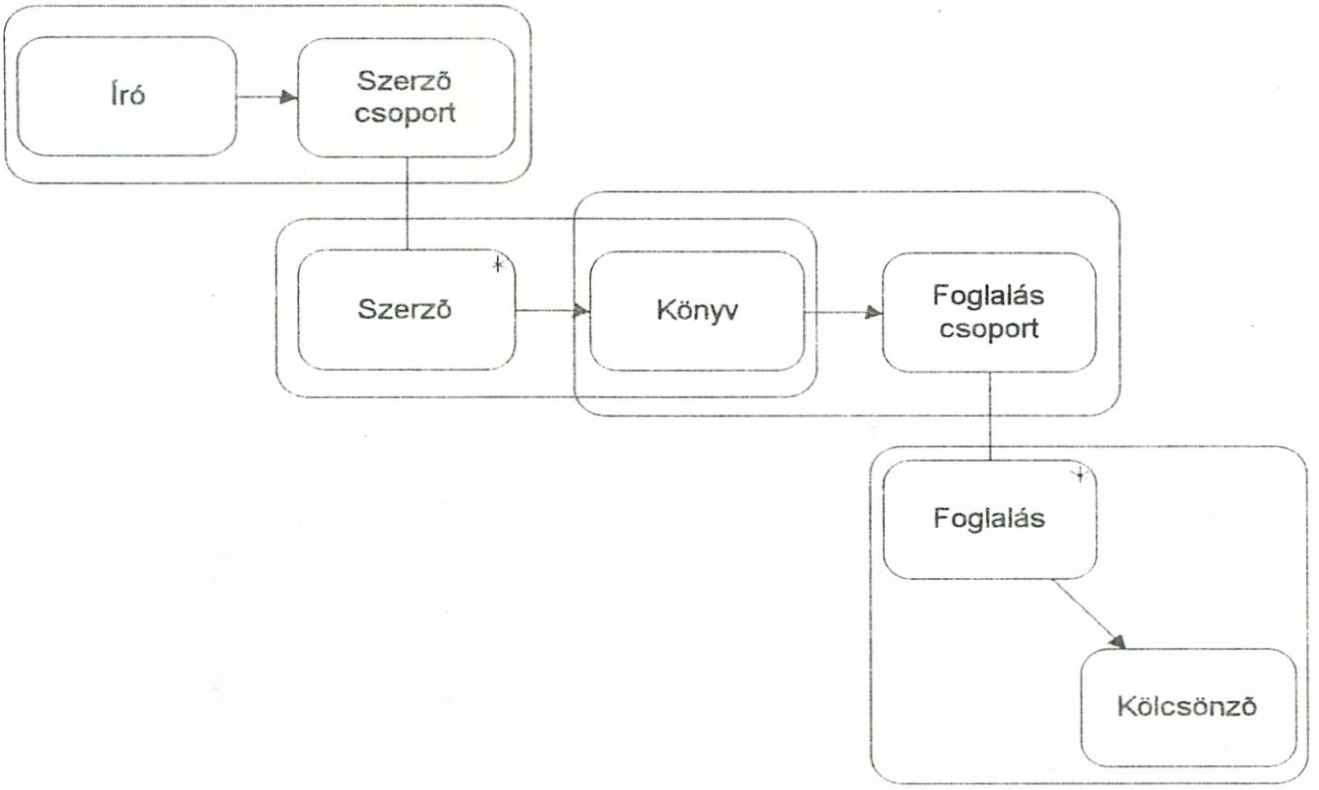
Ebből az I/O szerkezetből a lekérdezés outputját egy igen egyszerű művelettel kapjuk meg: elhagyjuk róla az inputot. Egyúttal még egy lépéssel tovább megyünk, és az új diagramon a dobozokat éles sarkúra alakítjuk, mert a végén a feldolgozás diagramján is erre lesz szükség. Az így átalakított diagram a 86. ábrán látható.


Foglalás író szerint
Író neve
Könyvek
*
Könyv
Könyv címe
Foglalások
Foglalás dátuma
Kölcsönző adatai

86. ábra

3.) Az elérések csoportosítása az elérési úton

A következő lépésben az elérési út diagramján végzünk átalakításokat. Az ún. 1:1 összefüggésben levő elemeket - tehát amelyek nyíllal kapcsolódnak egymáshoz - egy-egy csoportba foglaljuk, ahogyan ez a 87. ábrán látható.

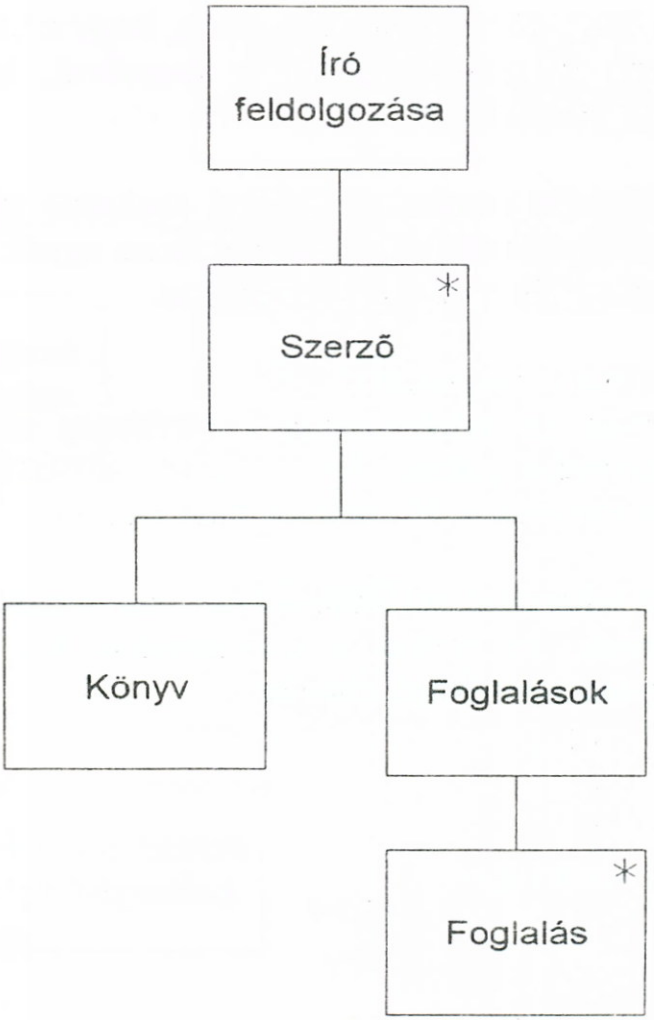


87. ábra

4.) Átalakítás Jackson-szerkezetté

Ezekből a csoportokból kell most "kihámoznunk" egy feldolgozási szerkezetet, amely már valódi Jackson-struktúraként ábrázolandó. Ezt csak úgy lehet helyesen elvégezni, ha gondolkodásunkat át tudjuk állítani a lekérdezési útról a feldolgozás menetére. E példa esetében az eredményt a 88. ábra mutatja, amelyhez mindenképpen magyarázatokat kell adni.

Úgy gondolom, hogy csak a diagram középső sávjában levő két csoportnak az átalakítása az, amihez részletesebb magyarázatra van szükség. Az elnevezések, amelyek a Jackson-struktúrán megjelentek, nyilván nem követhetik közvetlenül a lekérdezési úton levő elemek nevét, hiszen egy-egy csoportban több, különböző elem is van. A választott elnevezések akkor jók, ha már igyekeznek kifejezni azt, hogy itt feldolgozásokról - pontosabban feldolgozási lépésekről - van szó.



88. ábra

Az író adatainak feldolgozása alatt mindenképpen egy iteráció következik, ez már a csoportosításból is egyértelmű. Gondot okozhat viszont a két, egymást átfedő csoport értelmezése, ill. átalakítása Jackson-struktúrává. Abból indulunk ki, hogy minden SZERZŐ előfordulás feldolgozás maga után vonja egy KÖNYV előforduulás, valamint egy FOGLALÁS csoport feldolgozását, mégpedig ebben a sorrendben, ezért tehető szekvenciába ez a két feldolgozási elem.

5.) A két adatszerkezet egyesítése

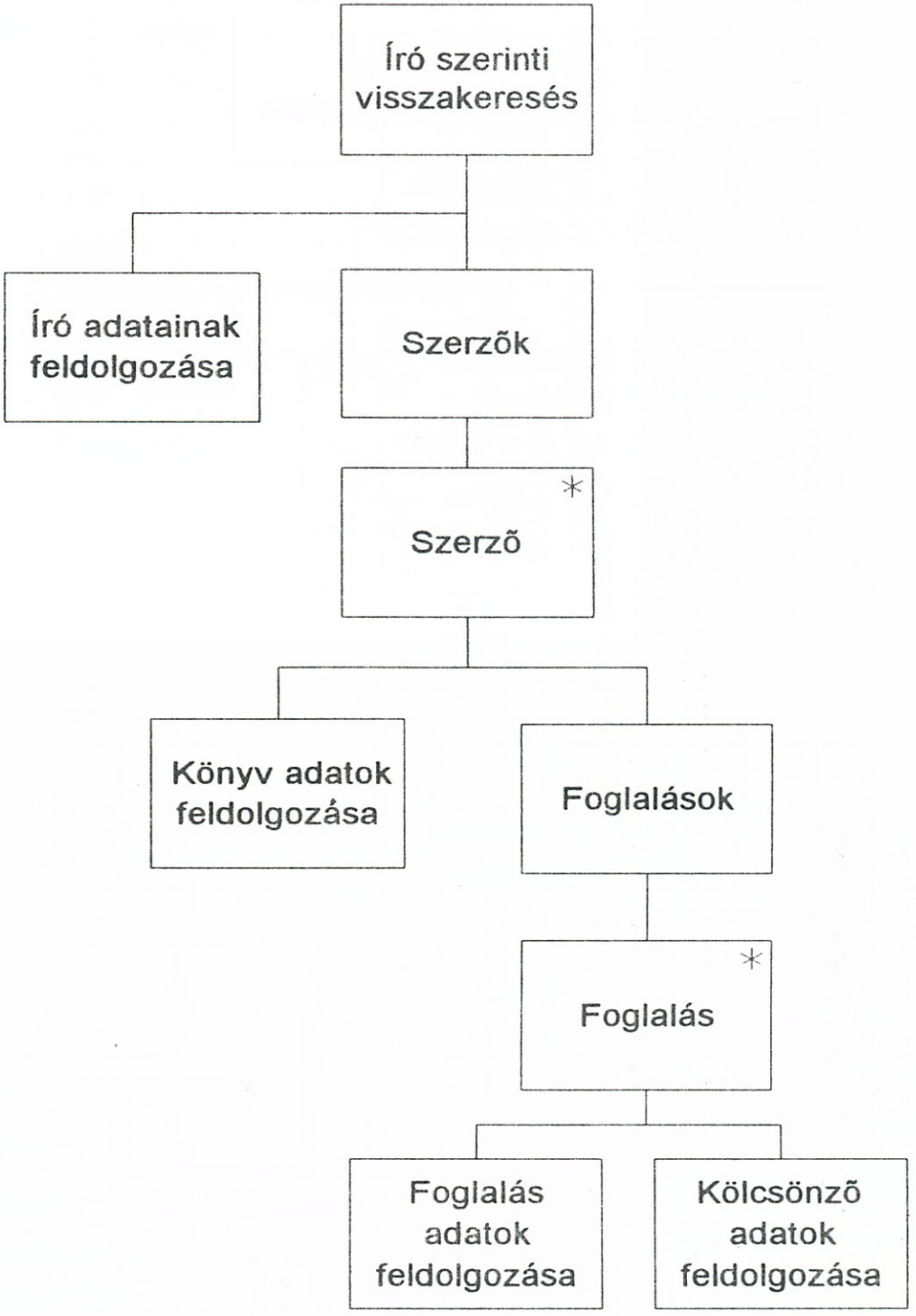
Mivel az I/O szerkezetből meghatároztuk az output iránti tartalmi igényt, az elérési út bejárását pedig input igényként fogjuk fel (aminek vezérelnie kell a lekérdezési úton való haladást), és mind a két feltételnek teljesülnie kell, ezért egyesítjük - mintegy "összetoljuk" - a két szerkezetet, amit előkészítettünk azáltal is, hogy mindkettő a feldolgozásokra érvényes Jackson-szerkezetek szabályai szerint készült. Az eredmény a 89. ábrán látható.

Az ábra értelmezésekor vegyük figyelembe azt is, hogy a szerkesztés szabályainak betartása miatt kellett új szintet nyitni a szerzővel, valamint a foglalással kapcsolatos feldolgozási elemeknél.

A viszakeresési feldolgozás szerkezete ezáltal csaknem teljesnek mondható. Az következő két lépésben az olvasási műveleteket és az egyes feldolgozási lépésekkel kapcsolatos feltételeket adjuk hozzá a diagramhoz.

5.) Műveletek feltüntetése a szerkezeten
Műveletek alatt itt csak az adatbázissal kapcsolatos műveleteket értjük, ami lekérdezés esetében olvasást jelent. Az olvasási műveletet minősíthetjük, hogy kifejezze, kulcs szerint történik-e, ill. milyen más alapon.

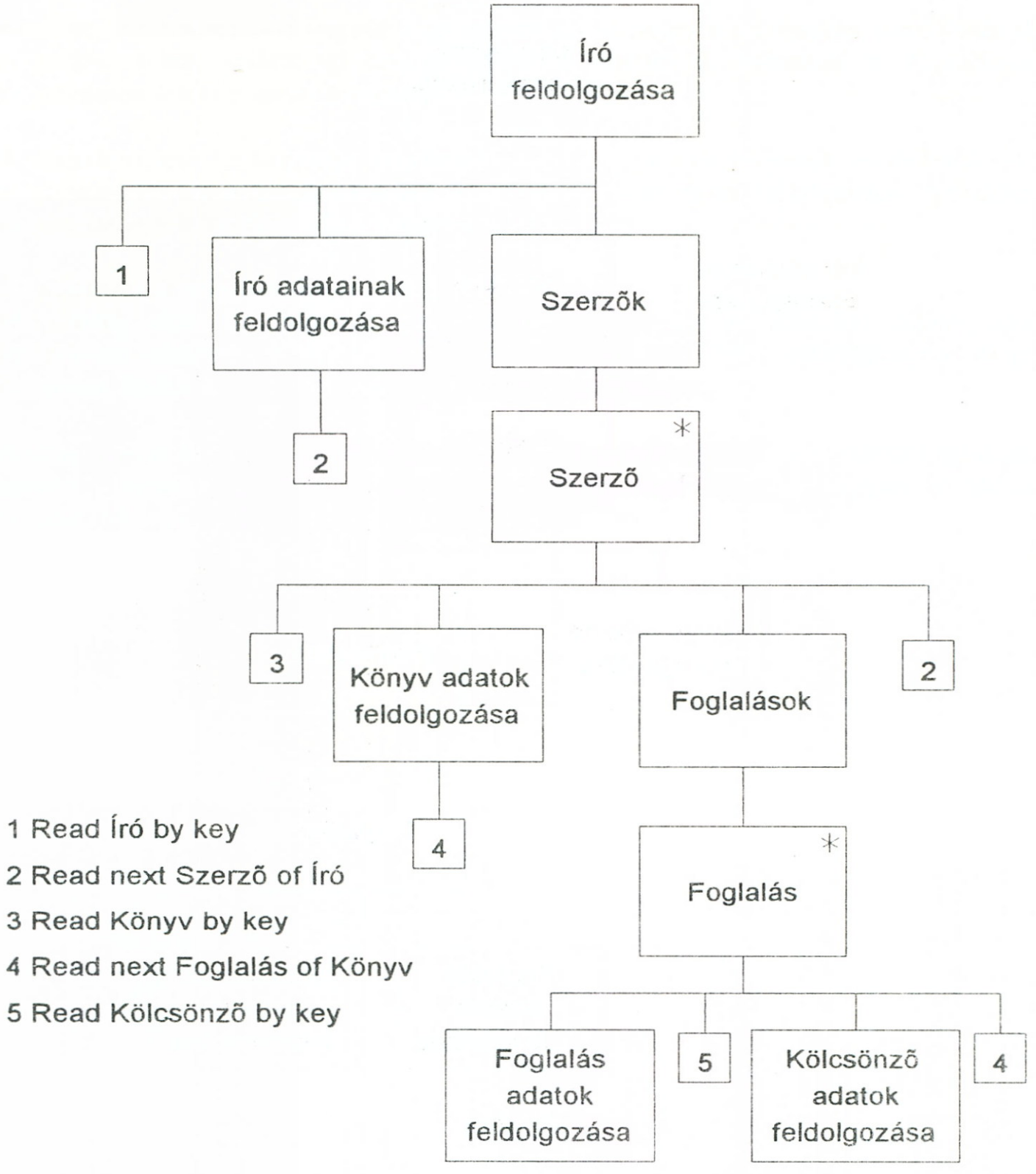
A diagramon ábrázolt lekérdezéssel kapcsolatos műveleteket jegyzékbe foglaljuk, amelyet besorszámozunk, és ezeket a sorszámokat kis négyzetekbe foglaltan hozzárendeljük a diagram vonatkozó részéhez.



89. ábra

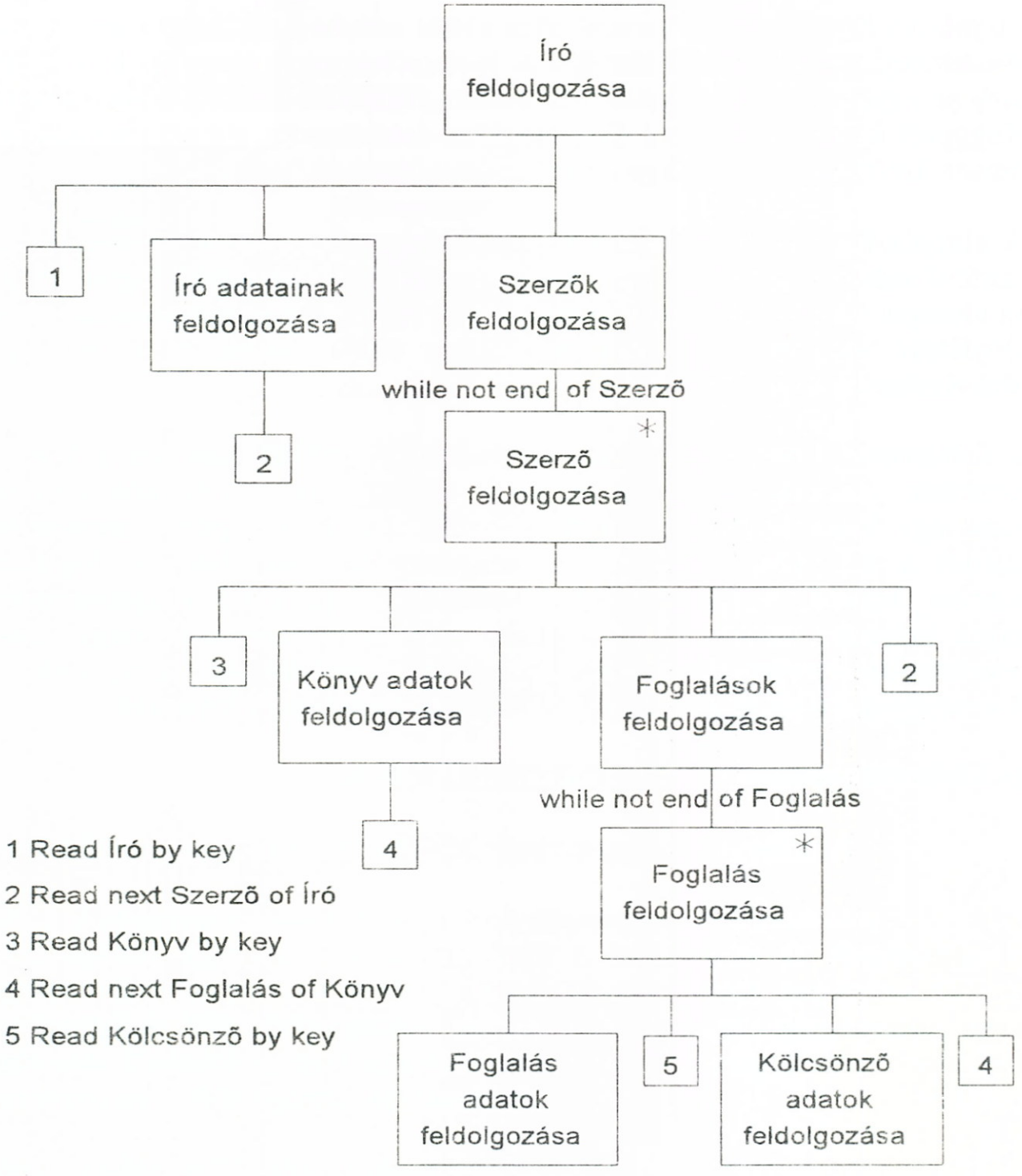
Magukat a műveleteket érdemes pszeudokód alakjában kifejezni, mert így egységesebb lehet a dokumentáció és esetleg később ez - szoftver környezettől függően - még automatikus kódgenerálásra is használható. A pszeudokód szintaktikája ennek megfelelően alakulhat.

A 90. ábra az olvasási műveleteket is feltünteti.


Író feldolgozása
1
Író adatainak feldolgozása
Szerzők
2
Szerző
*
3
Könyv adatok feldolgozása
Foglalások
2
1 Read Író by key
2 Read next Szerző of Író
3 Read Könyv by key
4 Read next Foglalás of Könyv
5 Read Kölcsönző by key
4
Foglalás
*
Foglalás adatok feldolgozása
5
Kölcsönző adatok feldolgozása
4

90. ábra

6.) Feltételek feltüntetése



91. ábra

A diagramokon minden iterációnál, ill. szelekciónál feltüntetjük azt a kifejezést, amely meghatározza az iteráció vagy szelekció feltételét. Esetünkben csak iterációk szerepelnek a diagramon, ennek megfelelően tüntetjük fel a "fájl vége" feltételt.

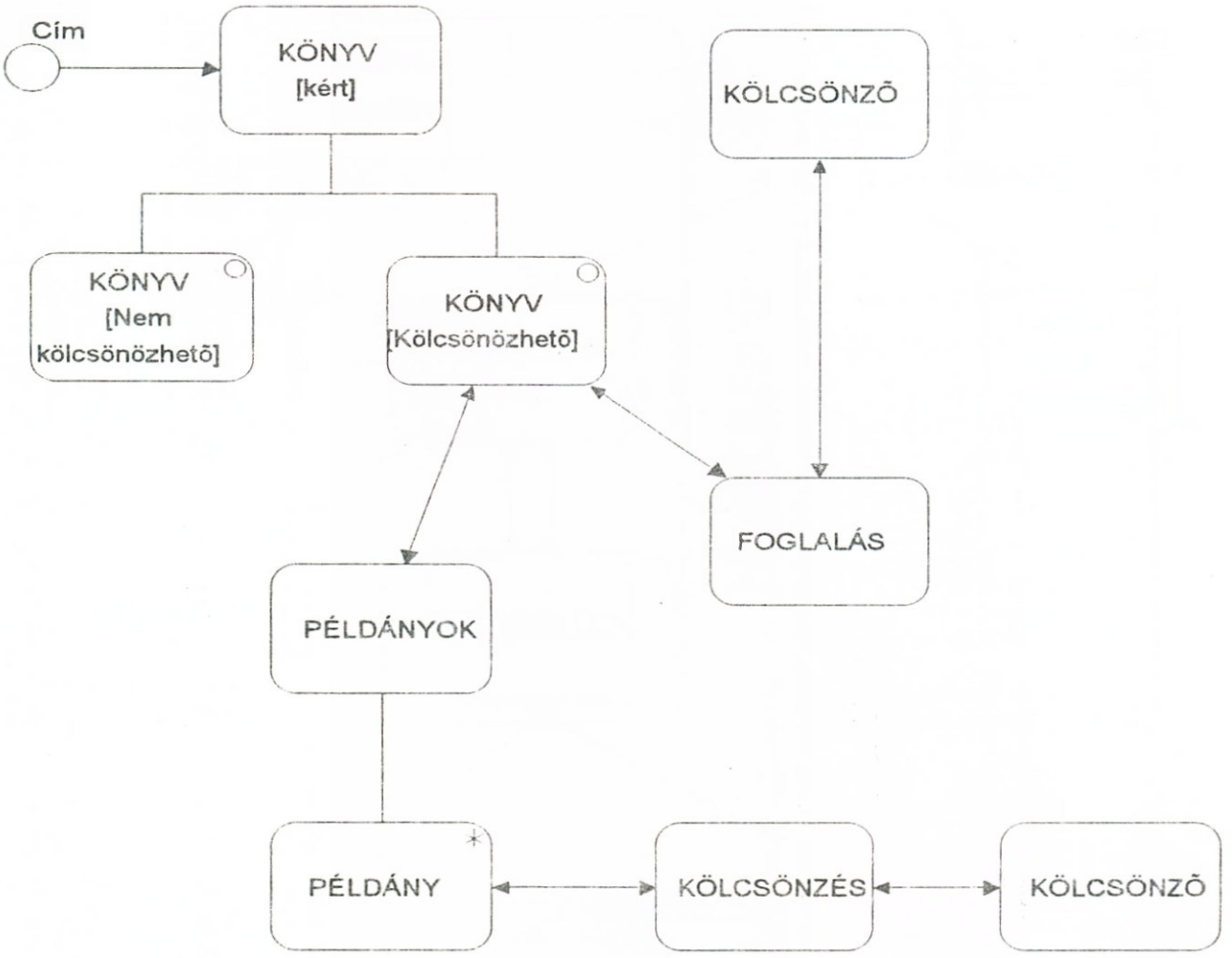
15.3 A karbantartó feldolgozások tervezése

A karbantartó feldolgozások tervezésekor abból indulunk ki, hogy - amint azt az egyed-esemény modellezésnél láttuk - a karbantartásokat események váltják ki. Ezeknek az *egyes* egyedekre gyakorolt hatását, valamint ezeknek a hatásoknak az összefüggését az egyedtörténeti diagramokkal modelezzük, míg az eseményhatás diagramon követjük végig az egy esemény által kiváltott karbantartásokat.

Ennek alapján logikus, hogy a karbantartó feldolgozás tervezése az eseményhatás diagramból indul ki, ezt mintegy a "karbantartás elérési útjaként" fogva fel. Itt az output szerepe csekély, mert nem lekérdezésről van szó. Végeredményben tehát az eseményhatás diagram fokozatos átalakítása révén jutunk el a feldolgozás Jackson-struktúrájához.

Ezt a folyamatot ismét egy példán keresztül tekintjük át, amelyet a könyvtári környezetből veszünk. A példa a 14.3 részben megismert könyv-foglalási példának egy kissé továbbfejlesztett változata. Az eredeti példa eseményhatás diagramját a 83. ábra mutatja. Ezt annyiban bővítjük, hogy a foglalást a kölcsönzési szándék meghiúsulásából vezetjük le, és ezzel tulajdonképpen a valóságos helyzetet közelítjük. *A foglalás előtt ellenőrizni kell, hogy van-e a könyvtárban az adott könyvből szabad példány, és ha igen, akkor az kikölcsönözhető, tehát új KÖLCSÖNZÉS előfordulást kell létrehozni.*

Az így kibővített esemény EHD-ja látható a 92. ábrán.

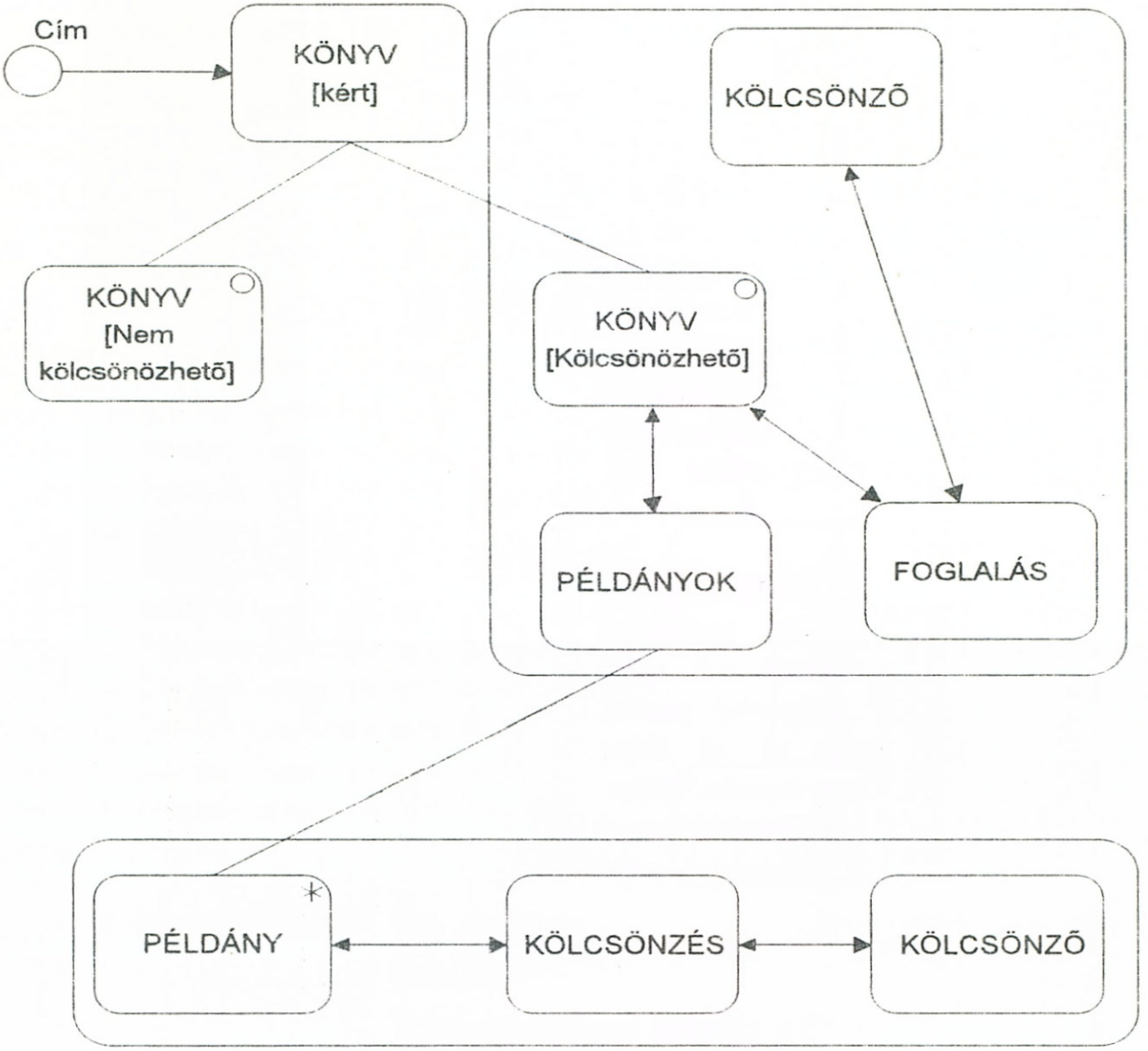


92. ábra

Ha az EHD megvan, akkor el lehet kezdeni az említett átalakítást.

1.) Az 1:1 megfeleltetésben álló hatások csoportjainak kialakítása
Ez lényegében ugyanaz a művelet, amit a lekérdezési út diagramján kellett elvégezni. Erre azért van szükség, mert így próbáljuk az egymással szoros összefüggésben álló feldolgozási műveleteket egy-egy feldolgozási egységbe összehozni. Tudomásul kell azonban venni, hogy a csoportosítás nem vezet el automatikusan a feldolgozási egységekhez.

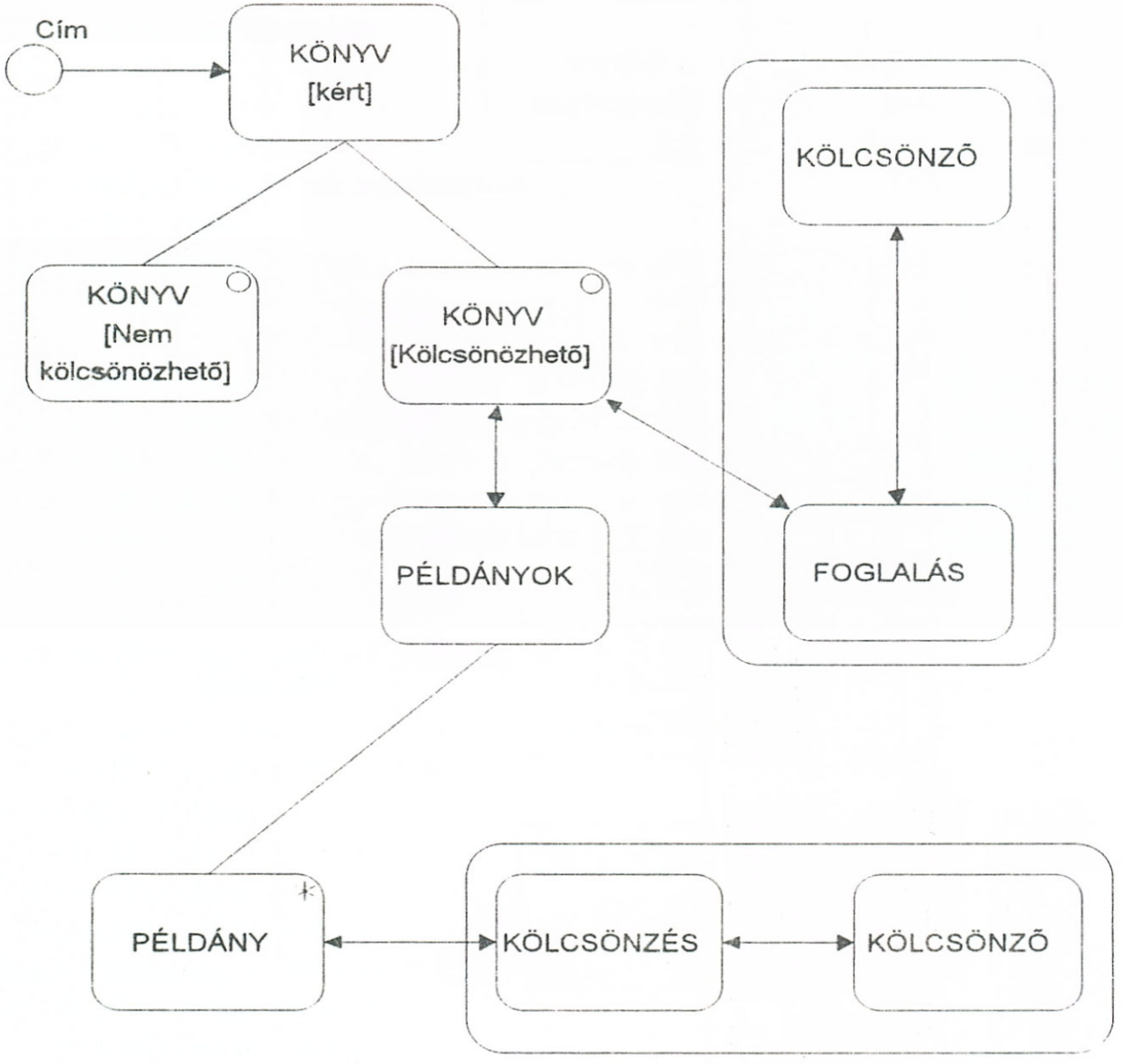
A csoportosítást a 93. ábra mutatja be.



93. ábra

Ez az első közelítésű csoportosítás bizonyos gondokat rejt magában. Először is nem fejezi ki azt, hogy foglalásra csak akkor kerül sor, ha a kölcsönzés sikertelen volt, aminek az az oka, hogy mindkét esetben 1:1 megfeleltetés van a kölcsönözhető könyv és a példányok csoportja, ill. a foglalás között. A második gond az, hogy bár a példány és a kölcsönzés között valóban 1:1 megfeleltetés van, ez csak az első megtalált példányra igaz, nem pedig mindegyikre.

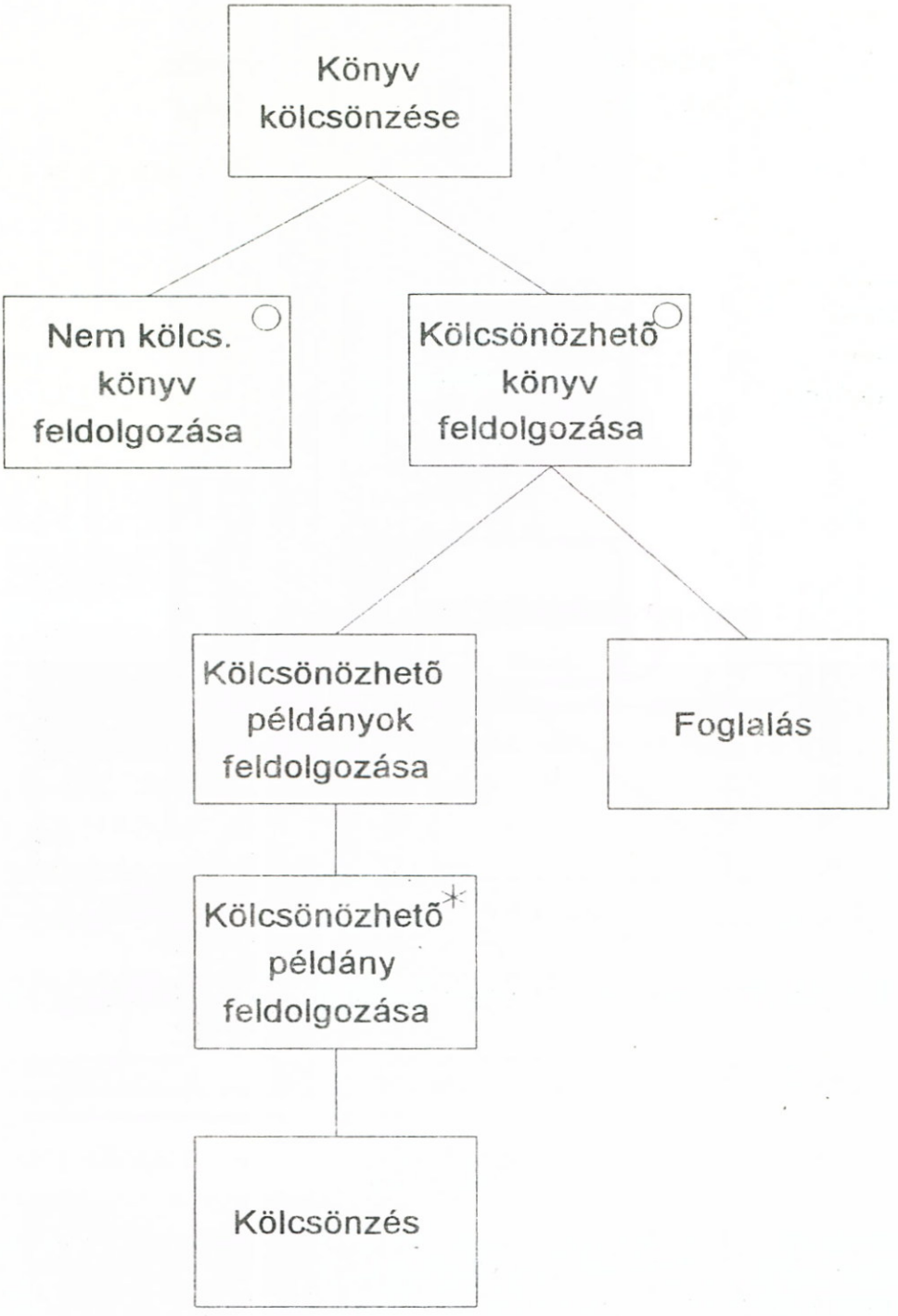
A gondokat feloldó csoportosítást látjuk a 94. ábrán.



94. ábra

2.) A feldolgozás Jackson-struktúrájának megrajzolása

Az előző diagram alapján ez már elég egyszerű feladat, de egyúttal fordulópontot is jelent a tervezési folyamatban, mert az eddigi - inkább az adatmodellhez kapcsolódó - szemléletről áttérünk a feldolgozásmodell-szemléletre, ami abban nyilvánul meg, hogy eltűnnek az egyedtípus nevek, saját nevet kapnak a bekeretezett csoportok és az egyed-szinbólumokat felváltják a feldolgozási egységek sarkos szimbólumai (95. ábra).



95. ábra

A továbbiakban a műveleteket és feltételeket kell meghatároznunk.

3.) A műveletjegyzék összeállítása

Azok a műveletek, amelyeket itt figyelembe kell venni, a - majdani - adatbázissal kapcsolatosak (logikai adatbázis feldolgozási műveletek). Főbb csoportjaik a következők:

- egyed adatainak olvasása
- állapotjelző ellenőrzése
- egyed létrehozása
- kapcsolat létrehozása
- egyed tulajdonságainak módosítása
- egyed eltárolása
- kapcsolat megszüntetése
- egyed törlése

Vitatkozni lehet azon, hogy a műveleteket leíró pszeudokódot milyen nyelven kell elkészíteni. A széleskörű érthetőség a magyarított pszeudokódot követelné meg, de figyelembe kell venni azt is, hogy a pszeudokódnak valamilyen módon valódi kóddá kell válnia, ami vagy programozási, vagy kódgenerálási munkát igényel. Mindkét esetben az angol alapú pszeudokód lehet a célszerű kiindulási alap, ezért példánkban is ez szerepel, meghagyva az egyedek magyar nevét.

Példánk esetében az alábbi műveletjegyzék állítható össze:

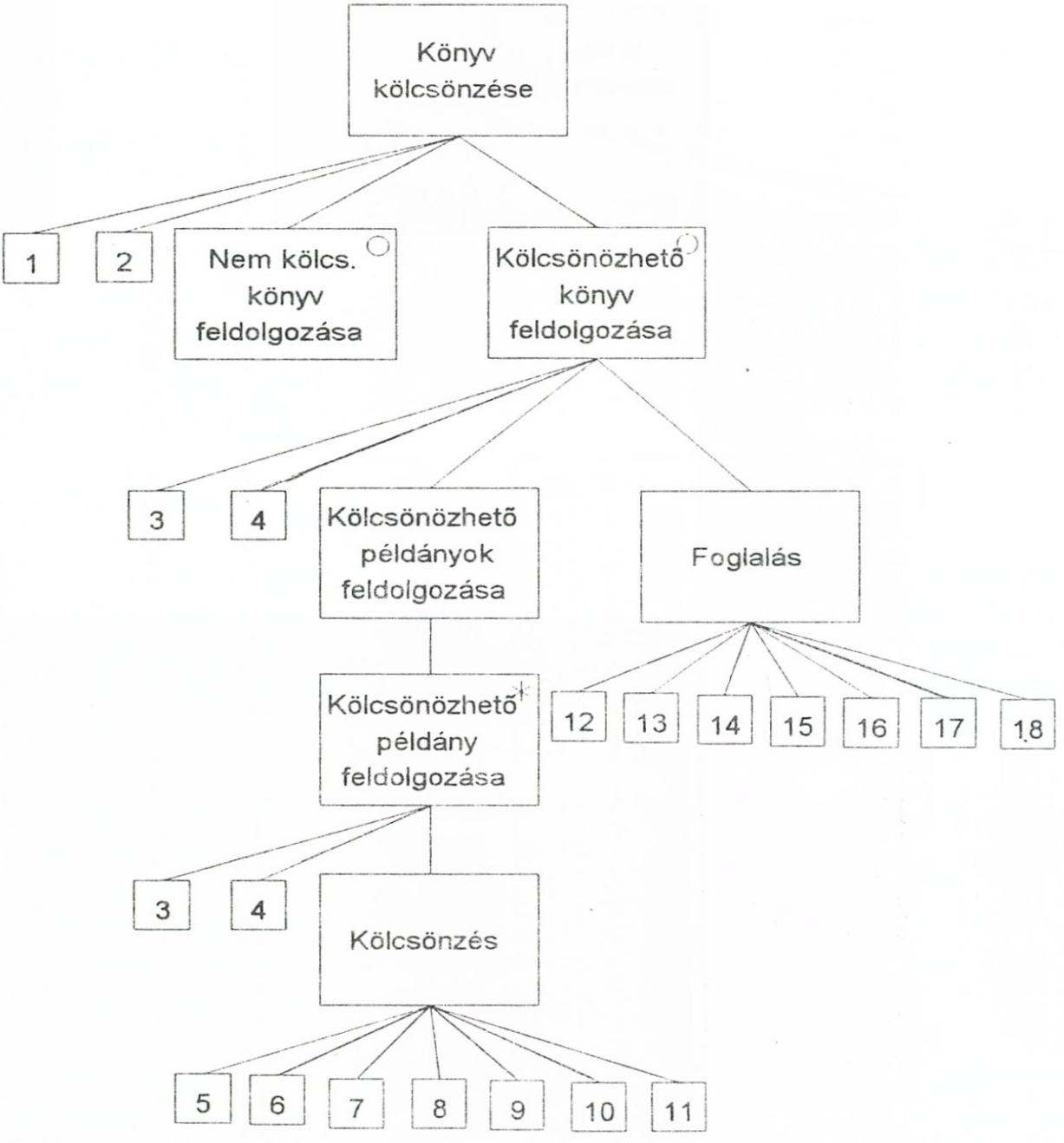
1. Read Könyv by key
2. Fail if SI of Könyv <> 1 or 3
3. Read next Példány
4. Fail if SI of Példány <> 2
5. Create Kölcsönzés
6. Store keys of Kölcsönzés
7. Store remaining attributes of Kölcsönzés
8. Set SI of Kölcsönzés to 1
9. Tie Kölcsönzés to Példány
10. Tie Kölcsönzés to Kölcsönző
11. Write Kölcsönzés
12. Create Foglalás
13. Store keys of Foglalás
14. Store remaining attributes of Foglalás
15. Set SI of Foglalás to 1
16. Tie Foglalás to Könyv
17. Tie Foglalás to Kölcsönző
18. Write Kölcsönző

Megjegyzés: az SI jelentése State Indicator, vagyis állapotjelző.

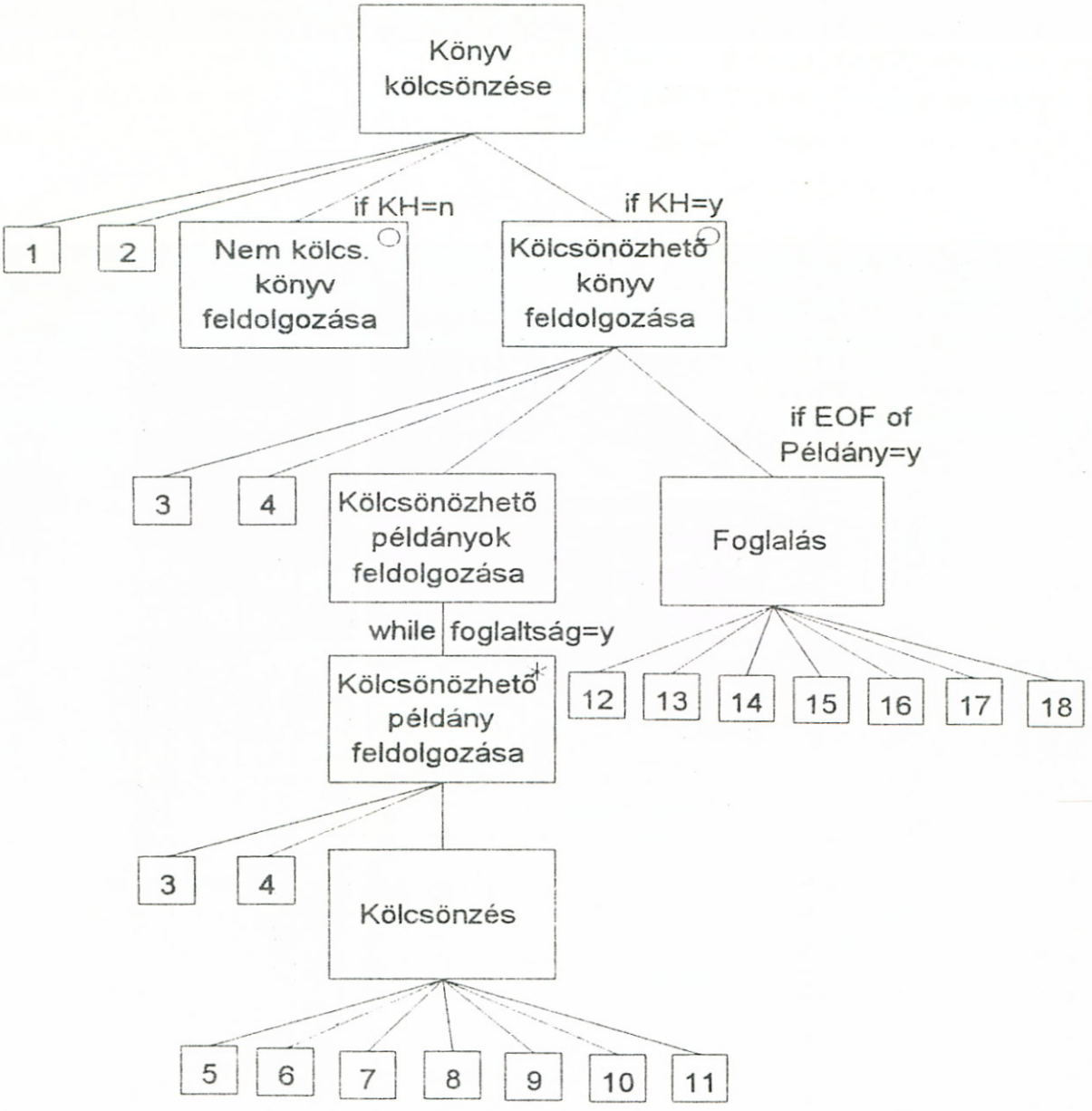
4.) A műveletek feltüntetése a diagramon
Ezt a lekérdezési feldolgozásokhoz hasonló módon, a műveletek sorszámának kis négyzetbe foglalásával és a négyzetnek a megfelelő feldolgozási egységhez való kapcsolásával történik. A megoldást a 96. ábra mutatja be.

5.) A feltételek feltüntetése
Ez is a lekérdezéseknél látottakkal hasonló módon történik, és a szelekció, valamint iteráció feltételeinek megadását célozza. Általában "while", ill. "if" kifejezésekről van szó, amelyeket a diagram vonatkozó részeire írunk rá.

A példánkra vonatkozó feltételekkel is kiegészített és véglegesnek tekintett diagramot a 97. ábra tartalmazza.


Könyv kölcsönzése
1
2
Nem kölcs. könyv feldolgozása
Kölcsönözhető könyv feldolgozása
3
4
Kölcsönözhető példányok feldolgozása
Foglalás
Kölcsönözhető példány feldolgozása
12
13
14
15
16
17
18
3
4
Kölcsönzés
5
6
7
8
9
10
11

96. ábra


Könyv kölcsönzése
1
2
if KH=n
Nem kölcs. könyv feldolgozása
if KH=y
Kölcsönözhető könyv feldolgozása
3
4
Kölcsönözhető példányok feldolgozása
if EOF of Példány=y
Foglalás
while foglaltság=y
Kölcsönözhető példány feldolgozása
12
13
14
15
16
17
18
3
4
Kölcsönzés
5
6
7
8
9
10
11

97. ábra

16. Fejezet
Dialógus tervezés

A mai modern információs rendszerek döntően on-line működésűek, tehát igen nagy szerep jut bennük a párbeszédeknek és a menüknek, ill. az ezek közötti mozgásnak, vezérlésátadásnak. Ezzel a tervezési problémával foglalkozik a dialógus tervezés technikája.

16.1 A dialógus tervezés célja

Ennek a tervezési munkának az eredményeképpen olyan specifikációhoz kell jutnunk, amely alkalmas a dialógusok fizikai szintű megtervezéséhez. Ez egyrészt jelenti a szükséges képernyők tartalmi terveinek elkészítését, másrészt pedig az ezek rendszerbe foglalásához szükséges menüszerkezetek és vezérlési szerkezetek kidolgozását.

Ahhoz, hogy a fenti célt meg lehessen valósítani, előbb el kell dönteni, hogy egyáltalán melyek azok a funkciók, amelyek dialógusokat igényelnek. Már a funkció meghatározásnál meg kellett adni olyan jellemzőket, hogy az adott funkció on-line vagy kötegelt, ill. a felhasználó által vagy a rendszer által indított. Azt mondhatjuk tehát, hogy azokhoz a funkciókhoz kell dialógust tervezni, amelyeket a felhasználó kezdeményez és on-line rendszerűek.

Említettem, hogy a dialógus tervezéssel kapcsolatban kell gondoskodni a menüszerkezet kialakításáról is. Itt két szempontot kell figyelembe venni: egyrészt a tartalmi összetartozást, másrészt azt, hogy mit igényelnek a felhasználók a munkájuk elvégzéséhez.

16.2 Dialógusok iránti igény meghatározása

Azt a folyamatot, amelynek eredményeképpen eldől, hogy mely funkciók igénylik dialógusok kidolgozását, a 98. ábra mutatja be.

98. ábra

Felhasználó katalógus elkészítése

Ez a munka már nagyon korán beindul a fejlesztés során. Tulajdonképpen akár a megvalósíthatóság tanulmányozása alatt elkezdhető — ha egyáltalán van ilyen munkaszakasz az adott projektben. A felhasználó katalógus igen egyszerű felépítésű:

- megnevezi az adott környezetbeli felhasználókat
- leírja az általuk végrehajtott tevékenységeket

Felhasználói szerepek

A felhasználó katalógus elkészítésének tulajdonképpeni célja, hogy megalapozza az ún. felhasználói szerepek meghatározását. A felhasználói szerep fogja egy csokorba mindazokat a tevékenységeket, amelyeket a különböző konkrét felhasználók végeznek, vagy végezhetnek. Ez azért fontos, mert a dialógusoknak e tevékenység csoportok végzését kell elsődlegesen szolgálniuk, tehát közvetlenül ezekhez rendelendők, a konkrét felhasználókhoz pedig csak közvetve. Ez a csoportosítás hivatott azt is megakadályozni, hogy esetleg többször is megtervezzenek egy projektben egy dialógust különböző felhasználók számára.

Nagy általánosságban úgy határozhatjuk meg a felhasználói szerepet, mint azon felhasználók csoportját, akik tevékenységi körében nagyfokú az átfedés. További szempontok, amelyek segítségünkre lehetnek a szerepek meghatározásában:

- munkaköri leírások azonossága, vagy hasonlósága,
- olyan felhasználók, akik ugyanazokkal a környezeti elemekkel (ld. AFD elemeinél) vannak kapcsolatban.

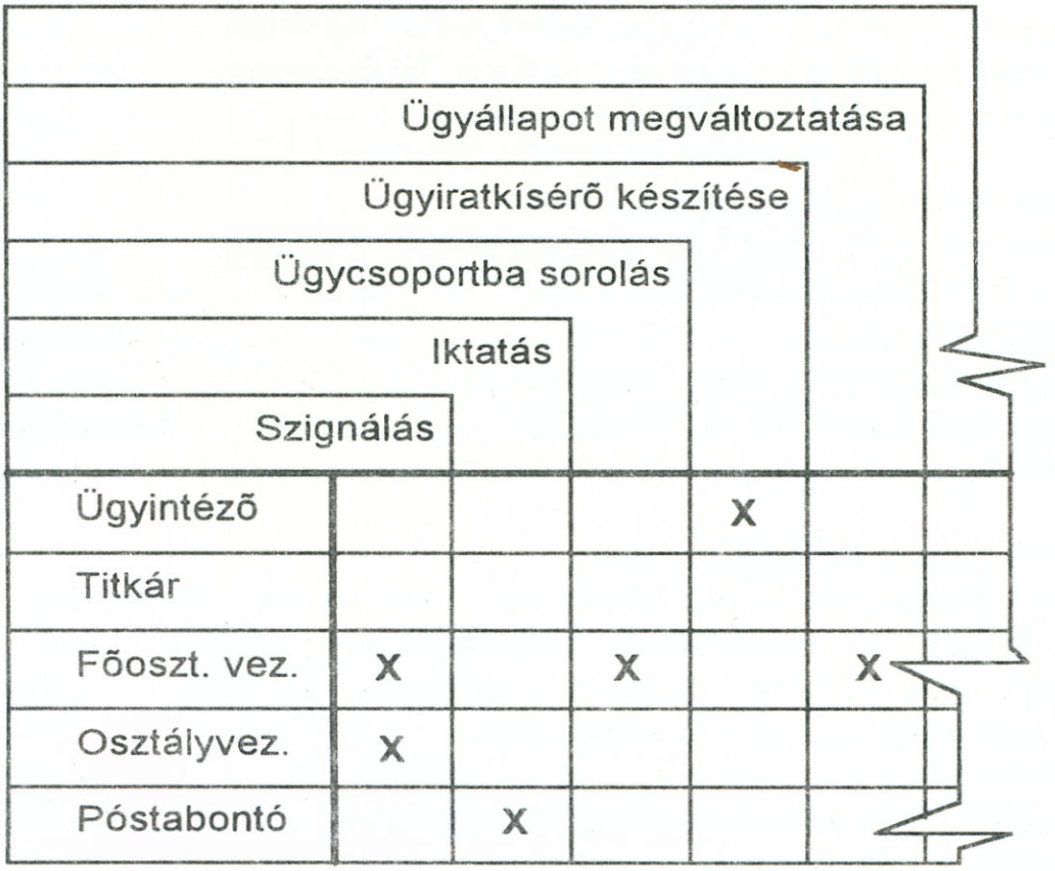
Arra is figyelemmel kell lenni, hogy azért nem feltétlenül tartozik ugyanabba a szerepbe minden felhasználó, aki azonos tevékenységet végez, mert eltérést okozhat pl. a különböző jogosultság.

A szükséges dialógusok meghatározása
Ezt a feladatot úgy lehet a legcélszerűbben megoldani, hogy egy mátrixot készítünk a felhasználói szerepekből és a végrehajtandó on-line funkciókból. Ahol ebben a mátrixban kapcsolat van feltüntetve (x), ott dialógusra van szükség. Az természetesen lehetséges, hogy ezen dialógusok nem mind lesznek különbözőek, hiszen természetes módon előfordulhat, hogy különböző felhasználói szerepek azonos funkcióval állnak kapcsolatban és azonos adatokra van szükségük.

Kritikus dialógusok meghatározása
A kritikus dialógusok kiválasztása már nem szoros értelemben a dialógus tervezés, hanem a specifikáció prototipizálás szempontjából fontos. A "kritikusság" tehát ebből a szempontból mérlegelendő. Ennek megállapításakor a következő szempontokat lehet érvényesíteni:

- A felhasználók mit minősítenek kritikusnak a munkájuk szempontjából?
- Igen gyakran használt dialógusról van szó?
- Sok egyedtípust használ a dialógus?
- Nagyszámú tulajdonságtípusot érint a dialógus?
- Sok felhasználó használja az adott dialógust?
- A felhasználó szervezet alaptevékenysége szempontjából központi szerepe van az adott dialógusnak?

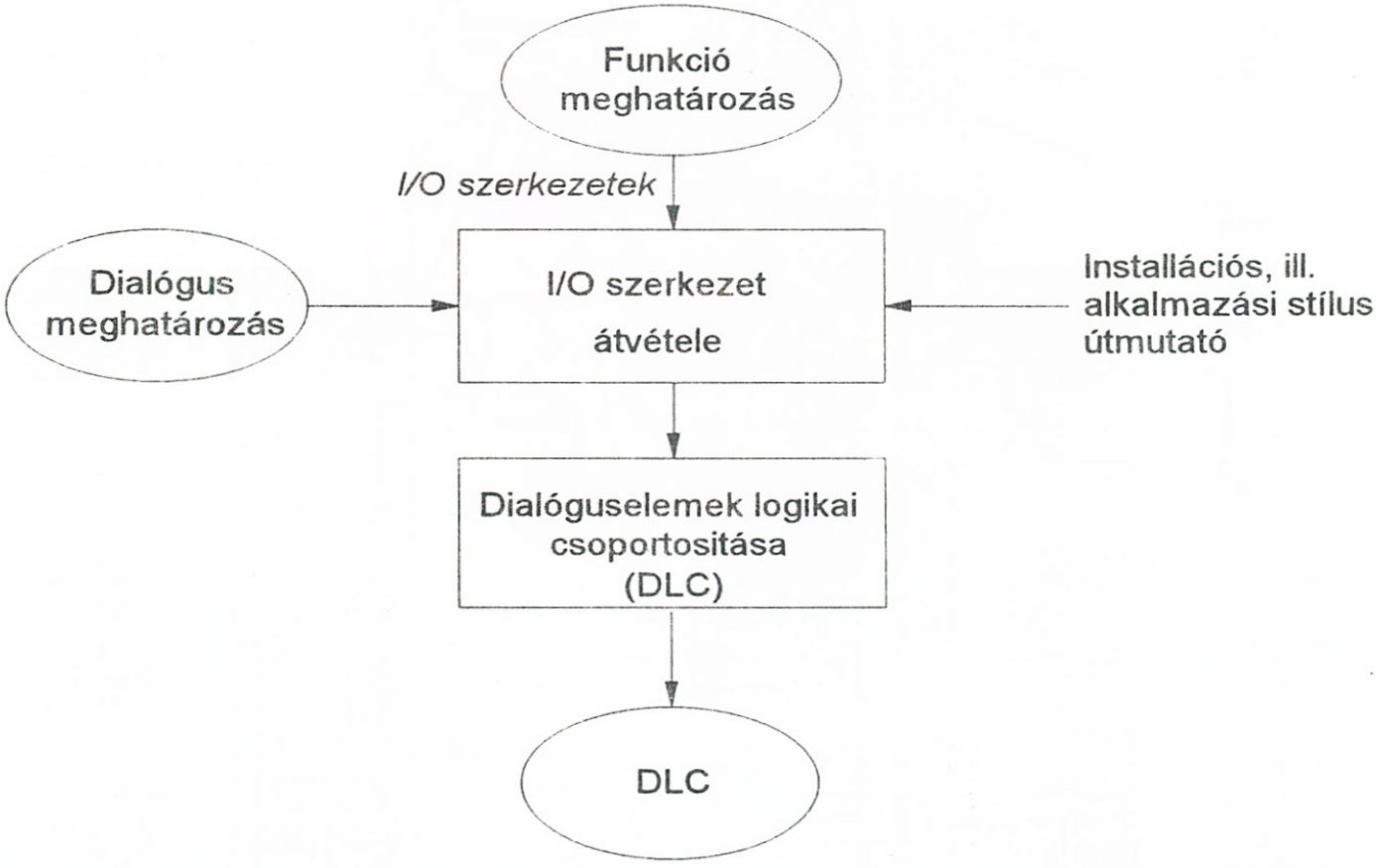
A kritikus dialógusokat a felhasználói szerep/funkció mátrixon tüntetjük fel úgy, hogy az ott szereplő megfelelő x-eket bekarikázzuk (ld. 99. ábra).



99. ábra

16.3 Dialógus tervezés

Ebben a pontban a szoros értelemben a dialógusok megtervezéséhez tartozó tevékenységekkel foglalkozunk. Ezeket mutatja be összefoglaló jelleggel a 100. ábra.



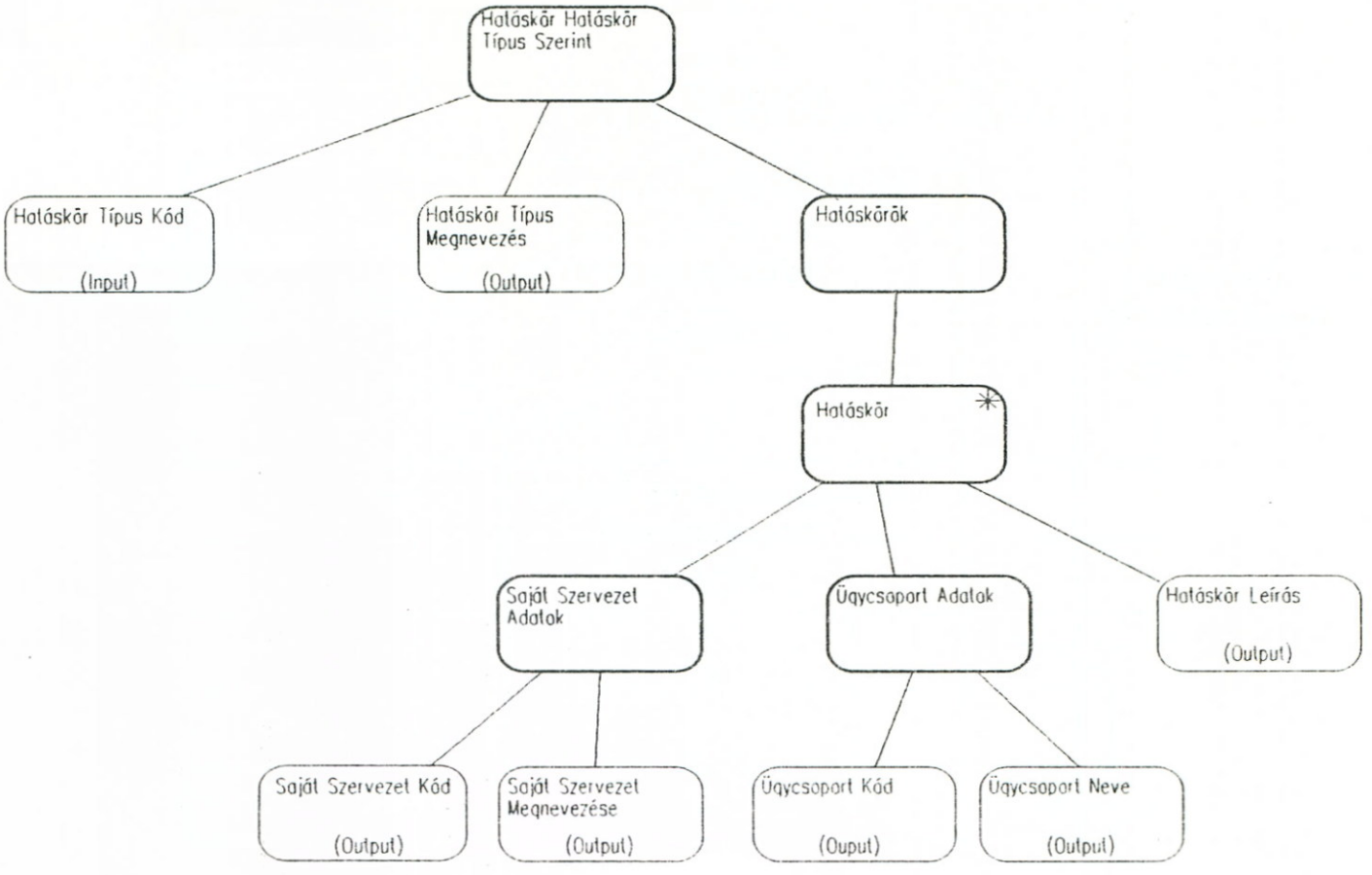
100. ábra

I/O szerkezet átvétele

A szoros értelemben vett dialógus tervezési munka teljes egészében a funkciókkal összefüggésben meghatározott I/O szerkezeteken alapul. Ezeket kell átalakítani dialógus szerkezetekké.

Ennek első lépéseként felülvizsgáljuk az eredeti I/O szerkezetet és az addig esetleg csak egy input tulajdonságtípust és az erre válaszul adott output tulajdonságtípus csoportot megtörjük úgy, hogy összetartozó input/output sorozat alakuljon ki — a tervezett párbeszédnek megfelelően.

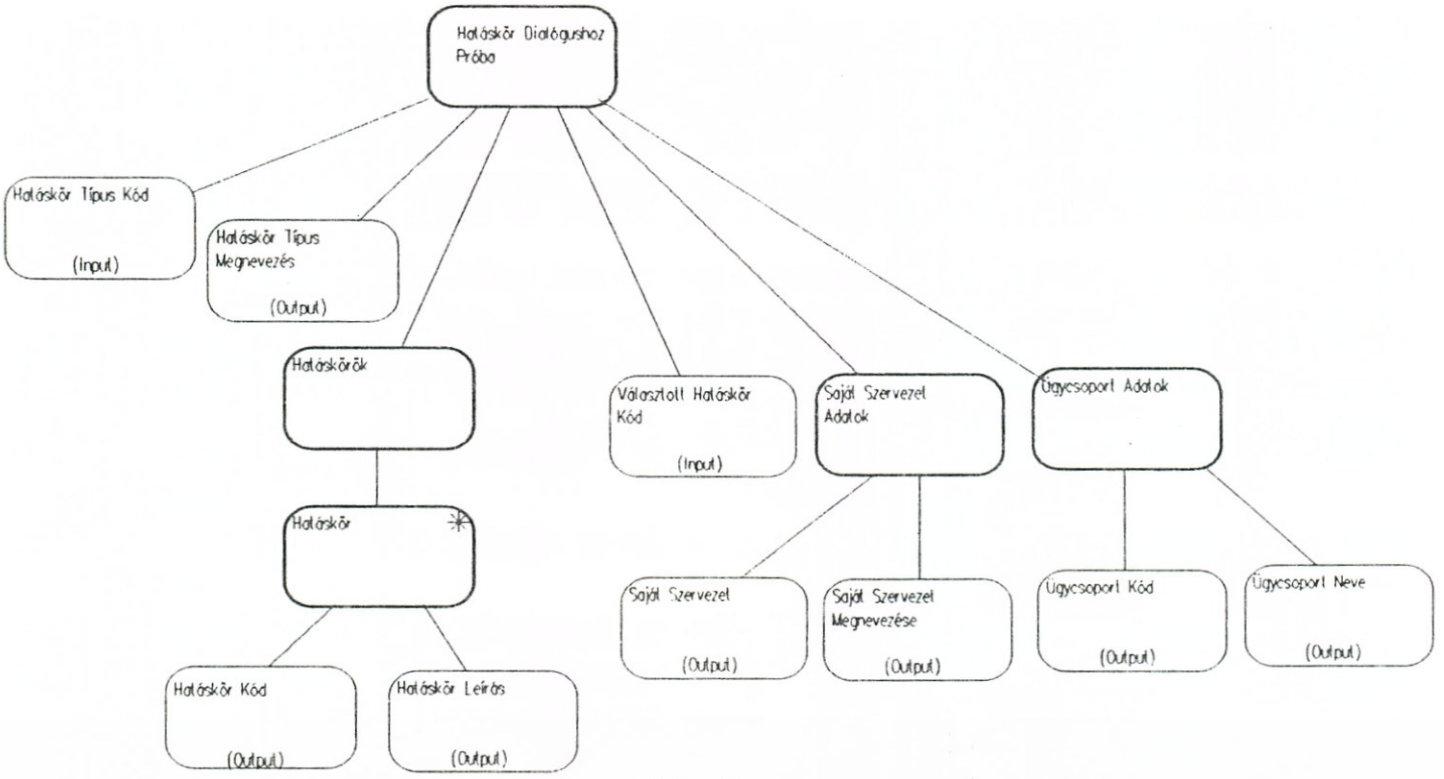
Vegyük példaként a 101. ábrán látható I/O szerkezetet, amely államigazgatási környezetből származik, és egy olyan funkció része, amely a hivatal egyes szervezeti egységeinek hatásköreit kérdezi le. Itt egy inputot, és egy sereg hozzá tartozó outputot látunk.



101. ábra

Ez a szerkezet akkor felelne meg, ha egy kötegelt feldolgozás lista-outputjának szánnánk. Ahhoz, hogy dialógusként működhessen, meg kell tördelnünk, ahogyan ezt a 102. ábra mutatja (az ábra, ill. a lap mérete miatt egy kis átrendezést is kellett csinálni).

Látható, hogy a terv alapján először a hatáskör típus kódját kell megadni (nem feltétlenül direkt módon), mire válaszul megkapjuk a hozzá tartozó hatáskörök jegyzékét. Innen kiválasztjuk a kívánt hatáskör kódját, mire válaszul megkapjuk annak a Saját Szervezetnek az adatait, amelynek a hatásköréről szó van, ill. az adott hatáskör ügycsoport szerinti besorolását.



102. ábra

Az így átalakított diagramon szereplő input, ill. output dobozokat a *dialógus elemei*nek nevezzük.

A dialógus elemek logikai csoportosítása

Ha a kialakított szerkezettel elégedettek vagyunk, akkor a következő feladat, hogy csoportokat (DLC) alakítsunk ki belőlük. Erre azért van szükség, hogy később erre alapozva határozhassuk meg a dialógus egyes részei közötti mozgásunkat, amit *navigáció*nak fogunk nevezni. A csoportok kialakításánál - bár nem fizikai tervezésről van szó - érdemes figyelembe venni a majdani képernyőn elhelyezendő karakterek maximális számát.

A csoportok kialakítása a legcélszerűbben a felhasználók bevonásával készülhet el. Ők tudják a legjobban, hogy a munka egyes műveleteit milyen sorrendben végzik, és hogy mi az, amit feltétlenül együttesen kell majd látniuk a képernyőn.

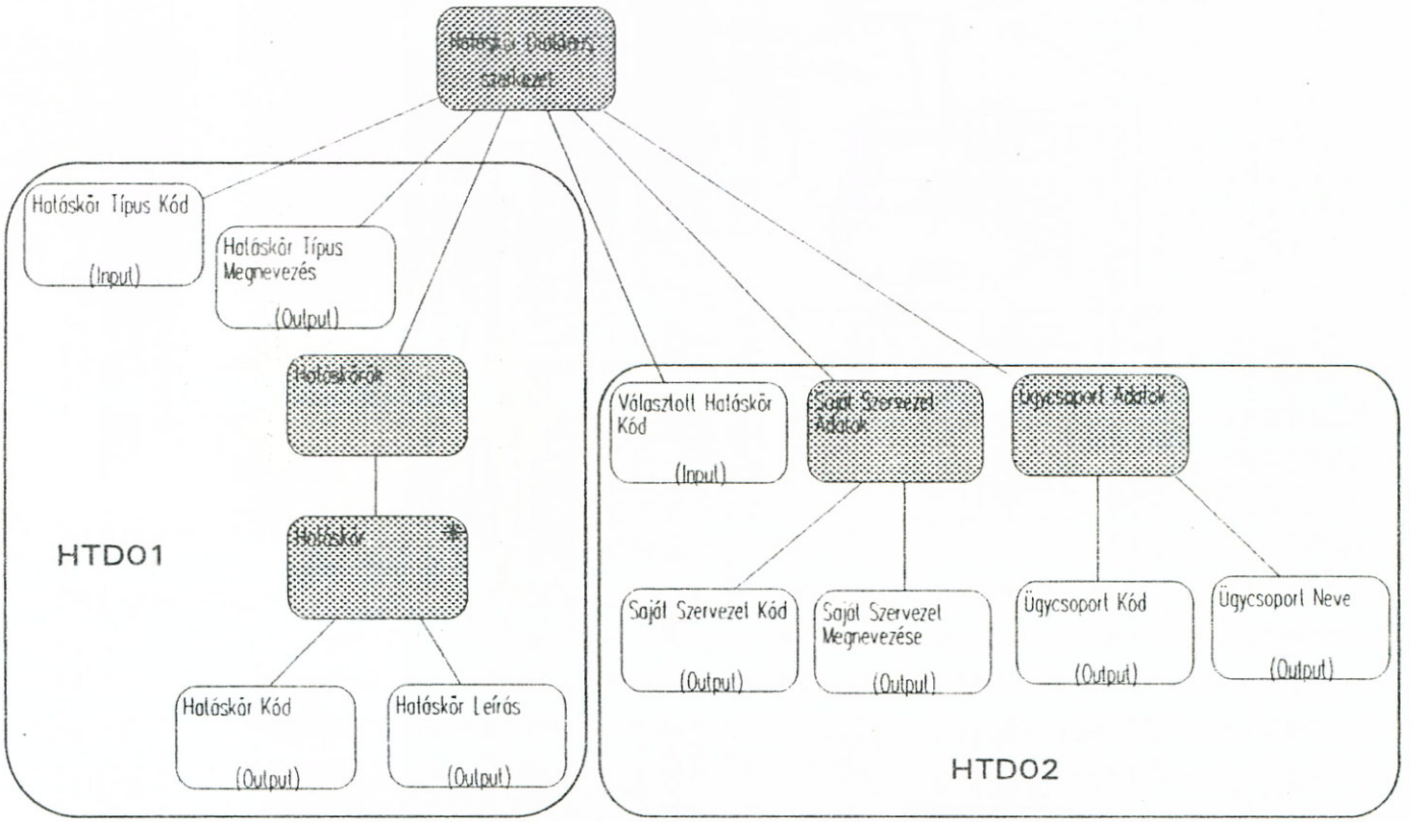
A DLC-k kialakításánál célszerű figyelembe venni, hogy

- tartalmazhatnak sorrendileg egymást követő elemeket,
- tartalmazhatnak olyan elemeket, amelyek szorosan összetartoznak,
- nem lehetnek úgy felépítve, hogy a szerkezeti diagramon a csoport elemei közé ékelődjenek nem a csoporthoz tartozó elemek,
- egy elem nem tartozhat két csoportba.

A leggyakoribb megoldás az, amikor egy inputot párosítunk egy csoportba a hozzá tartozó outputtal. Ez alól kivételt képeznek azok az esetek, amikor az adatok mennyisége - vagy más szempontok - ezt nem teszik lehetővé.

A kialakított csoportokat a diagramon bejelöljük és azonosítóval látjuk el (103. ábra).

Végül azt is meghatározzuk, hogy mely LCS szerepel kötelezően a teljes dialógus használatakor, és melyek azok, amelyeket "át lehet ugrani". Az előbbieket kötelezőnek, az utóbbiakat választhatónak tekintjük.



103. ábra

16.4 Menü tervezés

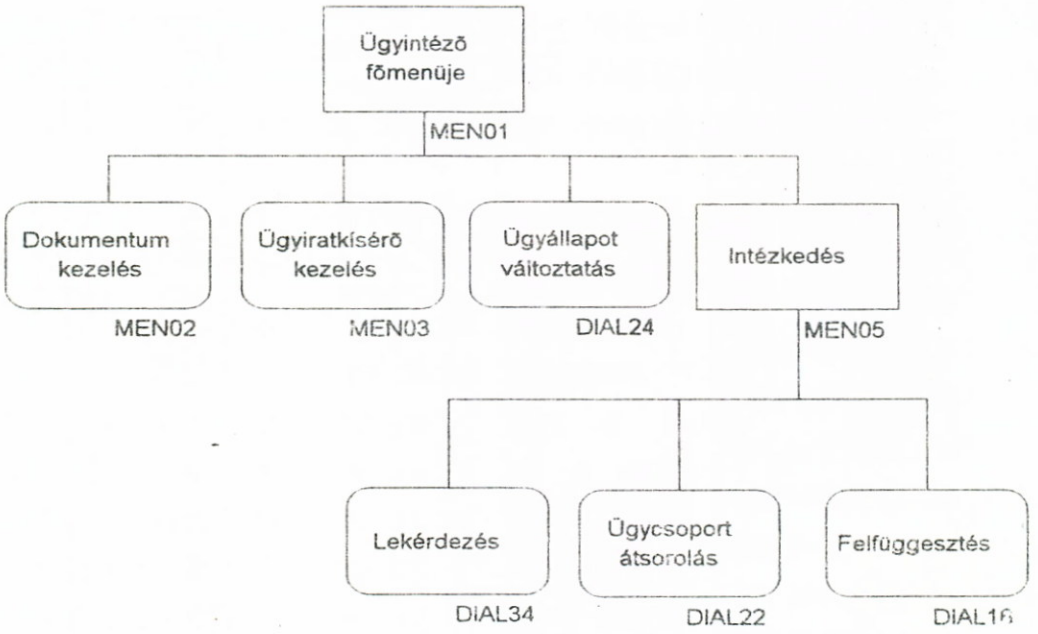
A menütervezés kiindulópontja a felhasználói szerep/funkció mátrix. Ebben a mátrixban minden "X" egy-egy dialógust jelez. Első lépésként úgy tekintjük, hogy az egy felhasználó számára szükséges dialógusoknak kell egy menüszerkezetbe kerülni, amit a mátrix soraihoz tartozó funkciók egy csokorba gyűjtése ad meg.

Az egy felhasználói szerephez tartozó dialógusokat azután hierarchikus szerkezetbe foglaljuk. Ez úgy történik, hogy magukat az eredetileg kifejlesztett dialógusokat csoportokba foglaljuk, majd a csoportokat magasabb szintű csoportokba. A dialógusok tehát ennek a hierarchiának a legalsó szintjén vannak.

Felmerül a kérdés, hogy hány szintje legyen ennek a hierarchiának, ill. hogy egy szint hány elemből álljon. Erre vonatkozóan nincsenek egzakt szabályok, de irányadó szempontok vannak, amelyeket célszerű figyelembe venni:

- A csoportosítás tükrözze vissza a felhasználói feladatok szabta követelményeket.
- Az egyes menüszintekről további menüszintekre vagy dialógushoz lehet eljutni.
- Az egyes szinteken levő elemek számának nem kell azonosnak lenni.
- Az összetartozó dialógusokat a tervezett rendszer AFD-inek tanulmányozásával könnyebb felfedezni.

Az esetleg korábban végrehajtott prototipizálás eredményeit is figyelembe kell venni. A menüszerkezetet egy olyan diagramon dokumentáljuk, mint amilyen pl. a 104. ábrán látható.



104. ábra

Irodalomjegyzék

CCTA	SSADM Reference Manual Version 4, Oxford, NCC Blackwell, 1990
MTA Információtechnológia Alapítvány	SSADM Strukturált rendszerelemzési és -tervezési módszer, Budapest, Miniszterelnöki Hivatal, 1993
E. Downs, P. Clare, I. Coe	Structured Systems Analysis and Design Method Application and Context, London, Prentice Hall, 1992
Malcolm Eva	SSADM Version 4: A User's Guide, London, McGrow-Hill, 1992
International SSADM User's Group	Delivering with SSADM (Conference Manual), Nottingham, 1993
dr. Bana István	SSADM - de facto módszertani szabvány, Budapest, IDG Computerworld Számítástechnika cikksorozat 1994 20-24 hét

JUDY fejlesztőrendszer

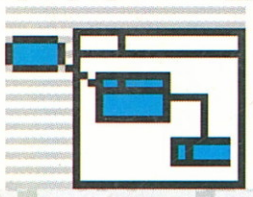
E könyv témái közt szerepel a CASE rendszerek elmélete (ld. 1.2 rész, 10. oldal). Ezek lényege tehát az automatikus rendszergenerálás, melyhez szélesebb értelemben hozzáértjük a helyzetfelméréstől a dokumentáció elkészítéséig tartó folyamat támogatását. E folyamatsorozat két fontos részét, a software-tervezést és -generálást automatizálja a JUDY fejlesztőrendszer.

A **JUDY** programrendszer alkalmazásával a programok széles körét elegendő csupán vázlatosan megtervezni; a software e terv alapján Turbo Pascal *nyelvű forrásprogramot ír.* A megvalósítás módja a CAD rendszerekhez hasonló; ott mérnöki munkát, itt pedig programot tervezünk és generálunk software-ekkel. A **JUDY** *programtervező és programot író* rendszer: a felhasználóval interaktívan, vizuálisan alakítja ki a terv finomított változatát. Elegendő egy vázlatos elgondolás, s a rendszerrel közösen, egyszerűen elkészített terv alapján a **JUDY** másodpercek alatt megírja a forrásnyelvű programot. A kapott program jellemzői: szintaktikailag hibátlan, strukturált; időre és tárra optimalizált, interaktív, áramkimaradástól védett; fejlett integrált környezetet, képernyős I/O-ot, aritmetikát, hatékony memóriakezelést tartalmaz.

A vizuális tervezés kevés programozási előismeretet igényel. A **JUDY** felhasználásával egy havi munkát egy nap alatt végezhetünk el, a software-készítéskor a lényegre koncentrálhatunk. A létrehozott programok más programnyelvekben - például Clipper, C, Cobol - is előnyösen használhatók. A rendszer *megvizsgálja a hardware-t,* s annak megfelelően működik. Ezt a tulajdonságot a generált programok automatikusan *öröklik,* anélkül, hogy erre gondot fordítanánk. Általánosan fogalmazva: *öninstallálók lesznek.* Így pl. automatikusan generálódnak az egérkezelési funkciók a leendő programba. Bővebb információval e könyv lektorához fordulhatunk: Pirkó József, 250-2620/213, valamint: 168-7044.

999,-

CASE a kézben ...



Teljes körű támogatás az SSADM-hez

Integrált elemzési-tervezési környezet keretében a módszer minden technikája rendelkezésre áll.

Fejlesztés kliens-szerver architektúrákra

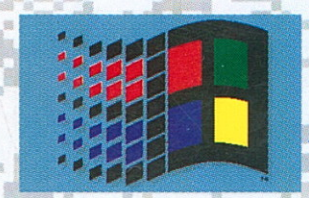
A kliensoldalon vizuális fejlesztő eszközökhöz kapcsolódik (PowerBuilder, SQLWindows, Visual Basic). Kódot generál és rekonstruál különböző szerveradatbázisokra (Sybase, SQL Server, Oracle, Ingres, Informix, SQL/400, SQLBase, DB2).

Komplett prototípuskészítési környezet

Képernyőtervezési és -bemutatási funkciói elősegítik a felhasználó szoros bevonására építő alkalmazásfejlesztést.

Biztosítja a csoportmunka hatékonyságát

Valódi többfelhasználós üzemmódban objektumtára (fejlesztési könyvtár) hozzáférés- és verzióellenőrzéssel is támogatja a fejlesztők munkáját.

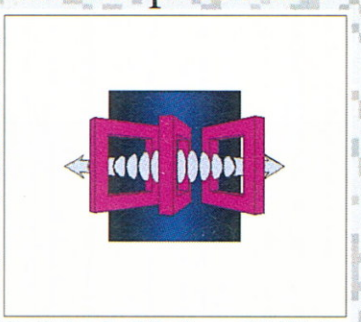


Windows-alapú használati felület

Garantálja az eszköz gyors megtanulását és produktív használatát.

Kerüljön az élre SSADM Engineer-rel!

IBIS Informatikai Kft., 1525 Budapest, Pf. 49
telefon: 1695-874, telefax: 1553-376



SSADM Engineer

Már ma a jövő szolgálatában