1. A programtermék minőségi mutatói

A szoftvertechnológia célkitűzése

- előírt minőségű programtermék,
- előre meghatározott határidőre,
- előre meghatározott költségen

történő előállítása.

Minőségi mutatók:

- programhelyesség,
- megbízhatóság, robusztusság,
- hatékonyság,
- pszichológiai bonyolultság,
- egyéb jellemzők, például kezelhetőség, újszerűség stb.

1.1. Programhelyesség

Ez a mutató a specifikáció és a program közötti kapcsolatot minősíti.

Tegyük fel, hogy adott egy feladat specifikációja a következő formában:

$X = \{x \mid p(x)\}$	bemenő adatok halmaza,
$Y = \{y \mid q(y)\}$	eredmény adatok halmaza,
$X \to Y$reláció	$\{(x, y) \mid p(x) \wedge q(y) \wedge r(x, y)\}$,

ahol p, q, r állítások!

Tegyük fel továbbá, hogy adott egy S program, amelynek programfüggvénye f_S! Ekkor azt mondjuk, hogy az S program a p, q, r specifikáció szerint helyes, ha

- végrehajtása befejeződik és
- $p(x) \wedge q(f_S(x)) \wedge r(x, f_S(x)) =$ `igaz`.

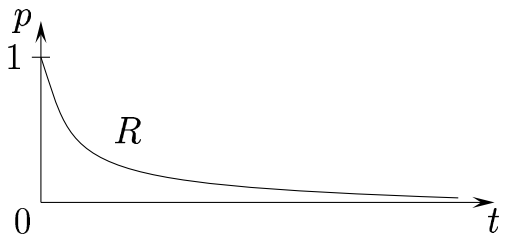
1.2. Megbízhatóság, robusztusság

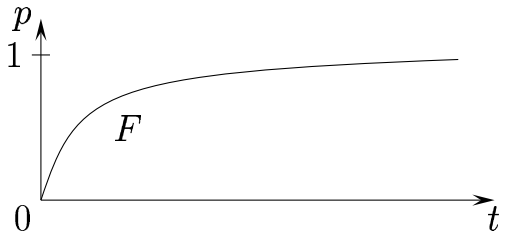
Ez a fogalom a programmal szemben támasztott követelmények és a program viszonyának minőségét fejezi ki. Annak valószínűségével mérhető, hogy a program nem nyújt hibás szolgáltatást.

Nem megengedett, bemenő adat ($x \notin X$) esetén a helyesség nem mond semmit, de ha a program megbízható, akkor ilyenkor sem nyújthat helytelen szolgáltatást (nem abortálhat).

Fordítva: a programunk megbízható, de nem helyes, ha olyan $x \in X$ bemenő adat esetén, amikor túlcsordulás lépne fel, akkor erre figyelmeztet, és nem hajtja végre a műveletet.

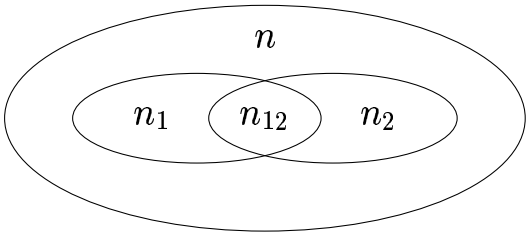
A megbízhatóság időben változó, amit jellemezhetünk a megbízhatósági függvénnyel R, illetve a meghibásodási függvénnyel F.





A megbízhatóság mértékét meghatározhatjuk a programban lévő rejtett hibák számának becslésével. Erre egy lehetséges módszer a következő.

Tegyük fel, hogy adott két azonos felkészültségű csapat a programban lévő rejtett hibák keresésére! A programban összesen n rejtett hiba van, amelyből az egyik csapat n_1-et, a másik pedig n_2-t talált meg, és ezek között n_{12} közös.



Ha feltesszük, hogy a két csapat azonos hatékonysággal dolgozik, akkor

$$\frac{n_{12}}{n_1} = \frac{n_2}{n} \quad \text{és} \quad \frac{n_{12}}{n_2} = \frac{n_1}{n},$$

ahonnan a megbízhatóság mértéke (a rejtett hibák számának reciproka):

$$\frac{1}{n} = \frac{n_{12}}{n_1 n_2}\ .$$

1.3. A program bonyolultsága

Ennek lényeges összetevője a program szerkezeti (architektúrális) bonyolultsága. A program akkor bonyolult, ha nehéz annak a működését megérteni.

Akkor könnyű egy program működését megérteni, ha

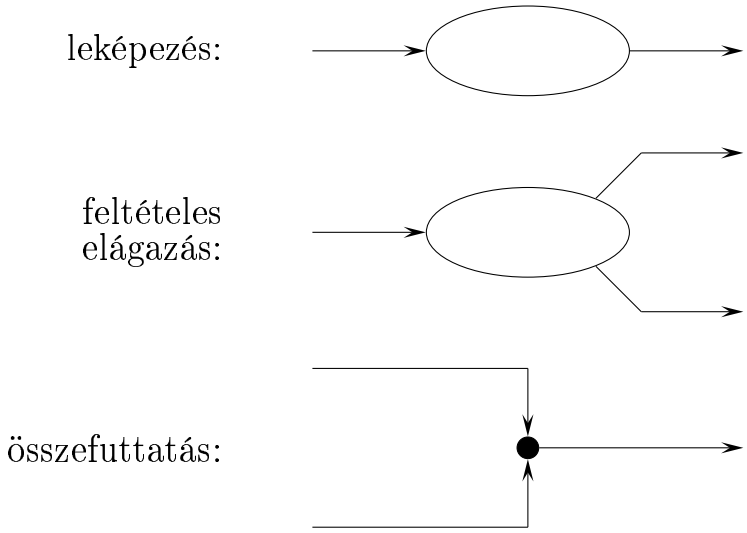
- egyszerű,
- az egymásra épülő absztrakciós szintek jól definiáltak,
- világosan elkülönül egymástól az egységek közötti kommunikációs felület és az egység szolgáltatásainak megvalósítása,
- az architektúra bővelkedik elegáns megoldásokban.

Felmerül a kérdés, hogy miként lehet mérni a program bonyolultságát. Ha megvizsgáljuk, hogy mivel lehet ezt becsülni, akkor két kézenfekvő lehetőség adódik:

- utasítások számával és/vagy
- döntések számával.

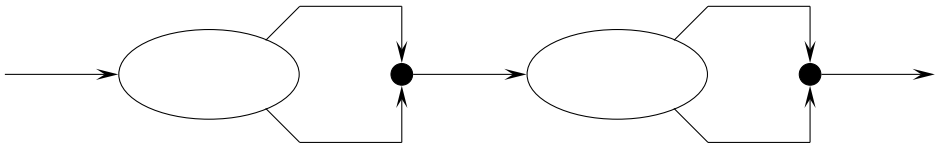
A döntések számának felhasználásán alapuló bonyolultságbecslésre mutatunk egy módszert a következőkben.

Tegyük fel, hogy a programban leképezések, feltételes elágazások (döntések) és összefuttatások szerepelnek, és minden csomóponton vezet legalább egy út a bemenő éltől a kimenő élig!

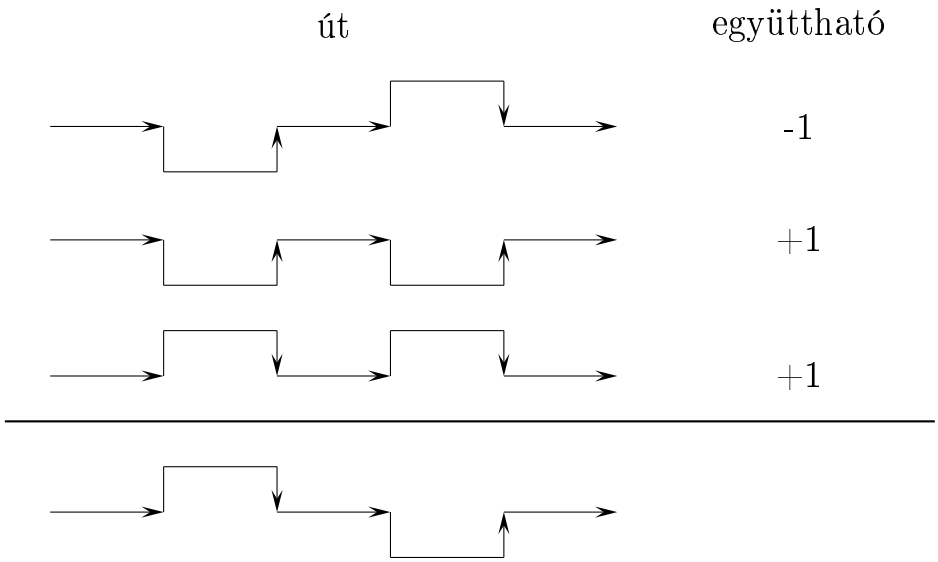


Ebben az esetben a programon átvezető, lineárisan független utak számának maximuma: $\pi + 1$, ahol π a programban lévő döntések száma.

Példa:



Bonyolultság: $\pi + 1 = 3$, mert:



1.4. Teljesítmény

Nagy rendszereknél lényeges szempont lehet a rendszer hatékonysága. Hiába működik egy rendszer hibátlanul, ha az egyes műveletek elvégzésének ideje nagyobb egy előre meghatározott értéknél. Ha az időkorlátot átlépi a rendszer, akkor esetleg használhatatlanná válik.

Néhány példa a teljesítmény fontosságára:

- A NASA kénytelen volt min. 8 hónappal később fellőni egy műholdat, mert az irányító szoftver válaszidejei elfogadhatatlanok voltak, és a műhold állapotát és telemetrikus adatait elemző rész is lassú volt. Hatás: az egész kutatás veszélybe került.
- Egy fehérnemű kereskedés az interneten keresztül akarta népszerűsíteni a tavaszi kollekcióját. Annak érdekében, hogy sokan látogassák meg az internetes oldalt, a Super Bowl alatt hirdette azt. Másfél millió ember jelentkezett be az oldalra, amelyen videókat is lehetett látni. A kimerítő előzetes tervezés és több szerver használata ellenére, a látogatók akadozó képet és hangot kaptak. Továbbá minimum 5 % el sem tudta érni az oldalt. Hatás: a vásárlói bizalom megrendülése.

- Egy nagyvállalat új változatot akart készíteni a számlázórendszeréről. Eredetileg 2 év alatt szerették volna befejezni a munkát. 7 év után, már a harmadik változat készült el, de egyik sem volt elég hatékony. A harmadik változat az eredeti változat CPU idejének 60-szorosát használta.

- Internetes brókercégek nem tudták fogadni az ügyfeleket az 1997 október 27.-ei árzuhanás után. Túl sok ügyfél próbált kapcsolatot teremteni, amit nem tudtak kezelni. Túl sokat kellett az ügyfeleknek várakozni, ha egyáltalán elérték a céget. Hatás: a befektetők százezer dollárokat vesztettek.

A rossz teljesítmény következményei:

- Megromlott kapcsolat az ügyfelekkel.
- Elmaradt üzletek. (Csúcsidőben nem tudjuk kezelni az igényeket.)
- Bevétel kiesés. (Akár büntetés fizetése is késői teljesítés miatt.)
- Túlzott erőforrásigény.
- Csökkent versenyképesség.
- A projekt kudarca.

A gyenge teljesítmény oka rendszerint a rossz terv, nem pedig a gyenge implementáció. Ez azt jelenti, hogy a szoftver készítésének korai fázisában kellene ezt kezelni.

Ennek ellenére elterjedt a következő felfogás.

„Tedd rendbe később" magatartás: A hatékonyságot hagyjuk figyelmen kívül a fejlesztés nagy részében. Csak akkor foglalkozzunk vele, ha a program már helyesen működik, és a terv jól kifejezi a rendszer működését. A változtatások úgyis jól behatárolhatók és kis területet érintenek.

A magatartás alapjai a következő legendák:

1. A teljesítménnyel nem lehet semmit sem kezdeni, amíg nincs valami végrehajtható és mérhető dolog.
2. A teljesítmény kezelése túl sok időt igényel.
3. A teljesítmény modellek összetettek és költségesek.

A legendák cáfolata:

1. Teljesítmény modellek használhatók a fejlesztés korai tervezési szakaszában előrejelzésre. Így kiértékelhetők a tervezési alternatívák.
2. A teljesítmény kezelése nem jelent automatikusan jelentős többlet időt. A befektetett munka függ a kockázattól. Magas kockázat esetén több ráfordítás szükséges, de ekkor a fejlesztés teljes ideje rövidül a felesleges zsákutcák elkerülésével.
3. Egyszerű modellekkel már képesek vagyunk a szükséges becslésekre.

Teljesítmény kezelési sikerek:

- Egy repülőjegy foglalásokat intéző iroda frissítette a tarifa megállapító rendszerét, hogy a legolcsóbb tarifát pontosabban állapítsa meg. Itt folyamatosan figyelembe vették a teljesítményt a fejlesztés során. Az eredmény teljesen pontos tarifák, és jobb teljesítmény.
- Egy jelentős biztosító társaság olyan rendszert tervezett, amely internetes elérést biztosított a saját és független ügynökök számára. Az első változatban jelentős mennyiségű kódot kellett volna a kliens gépekre letölteni. A teljesítmény modellezés során kimutatták, hogy még a lényegesen továbbfejlesztett kód letöltése az összes kliensre 3 napig tartott volna teljes sávszélességet használva. Ezért a tervet megváltoztatták, hogy kevesebb kódot kelljen letölteni, és a rendszer sikeresen működött.
- Egy esemény naplózó rendszer kezdeti követelményei között szerepelt, hogy az eseményeket a bekövetkezésük után max. 3 perccel kell egy online adatbázisba elhelyezni. A korai teljesítmény elemzés kimutatta, hogy ehhez 20 mainframe gép kellett volna (a legjobb szoftvert feltételezve). Ennek alapján a célokat újra fogalmazták, ami már egy ésszerűbb hardver konfigurációt eredményezett.

A teljesítmény kezelésének módjai:

1. reagáló,
2. aktív.

1. Reagáló:

- Legyen kész, és nézzük mit tud.
- Később tuningoljuk, most nincs időnk a teljesítménnyel foglalkozni.
- Nem tehetünk semmit addig, amíg nincs valami, amit mérni lehet.
- Ne izgulj. A szoftver eladói biztosak, hogy a termékük teljesíti az igényeinket.
- Teljesítmény? Arra való a 2. verzió.
- Majd veszünk egy gyorsabb processzort.
- Nincsenek teljesítmény gondjaink.

2. Aktív:

- A projektben van egy „teljesítmény mérnök”, aki felelős a teljesítménnyel kapcsolatos problémák felkutatásáért és közléséért.
- A projekt minden tagja ismeri ezt a személyt.
- Létezik a teljesítménnyel összefüggő kockázatok azonosításának és feloldásának kezelésére egy eljárás.
- A projekt rendelkezik teljesítmény kockázat kezelési tervvel.

Az aktív megközelítés egyik legújabb eredménye a teljesítményelvű szoftvertechnológia (*SPE* – Software Performance Engineering) [1]. Ennek lényege, hogy nem igényel bonyolult matematikai apparátust, az UML alapú objektumelvű tervezéshez illeszkedik.

[1] Connie U. Smith, Lloyd G. Williams: Performance Solutions – A Practical Guide to Creating Responsive, Scalable Software, *Addison-Wesley, Pearson Education, Inc.*, 2002 [510] ISBN-0-201-72229-1.

2. Teljesítményelvű szoftvertechnológia

1. A teljesítmény szempontjából lényeges felhasználói esetek azonosítása. Ezekhez az esetekhez forgatókönyvet kell készíteni.

2. A forgatókönyvek készítésének eszköze a szekvenciadiagram. Ennek érdekében a szekvenciadiagram jelöléseit ki kell egészíteni.

3. A szekvenciadiagramok alapján el kell készíteni a végrehajtási gráfokat.

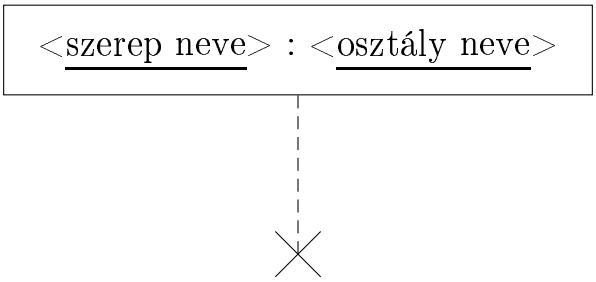
4. A végrehajtási gráfok alapján meghatározhatjuk az optimális, a legrosszabb és az átlagos eset futási idejét.

5. Ennek ismeretében tudunk válaszolni arra a kérdésre, hogy elég hatékony-e a rendszer.

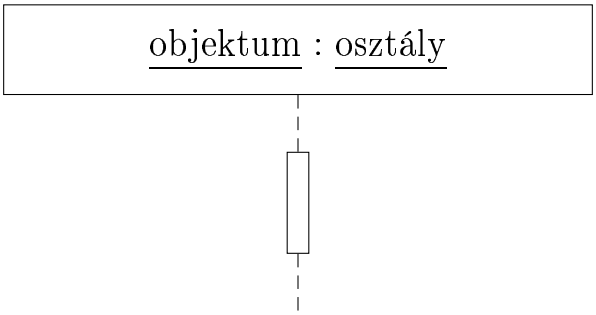
6. Arra is lehetőség van, hogy megmondjuk, létre lehet-e hozni a megadott paraméterekkel rendelkező rendszert. Ha nem, akkor jelezhetjük a megrendelőnek ezt, és módosítani lehet az elvárásokon, mielőtt még akár a fejlesztő, akár a megrendelő lényeges ráfordításokat hajtott volna végre.

2.1. Szekvenciadiagram (ismétlés)

Osztályszerep, életvonal:

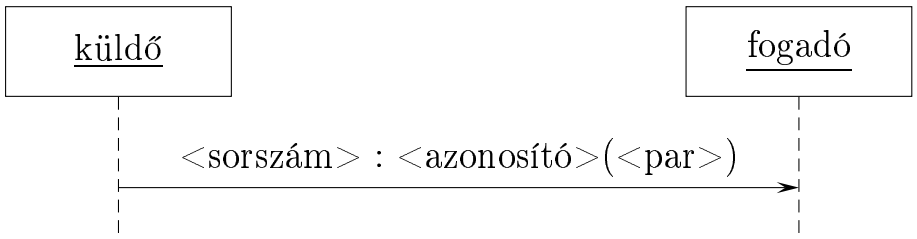


Aktivációs életvonal:



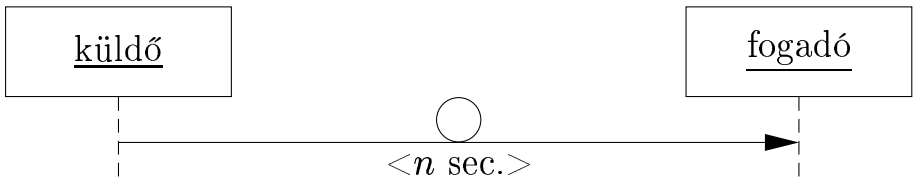
Üzenetek:

egyszerű:



szinkronizációs:

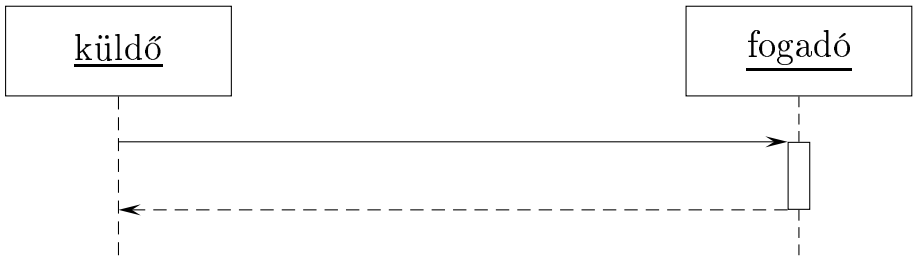
időhöz kötött várakozás:

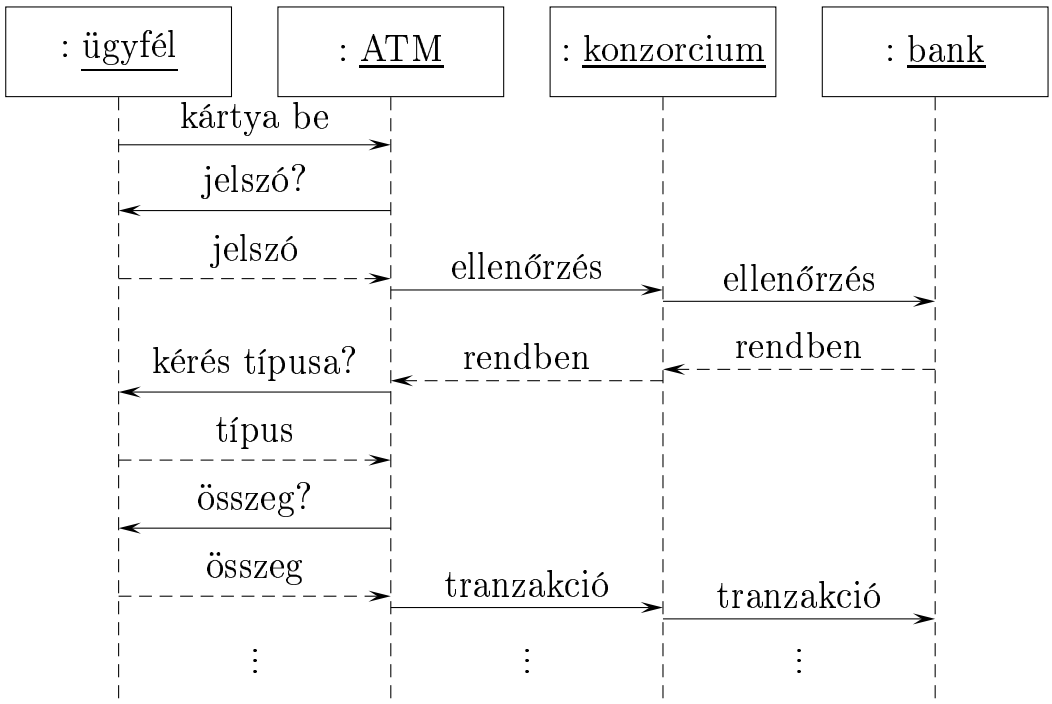


randevú:

aszinkron:

visszatérési (válasz):

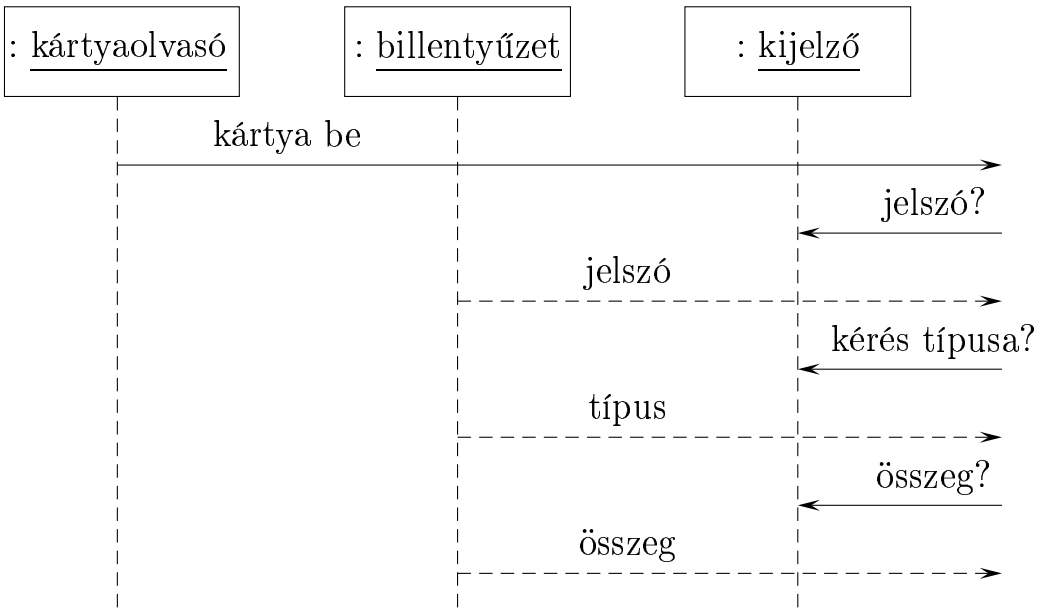


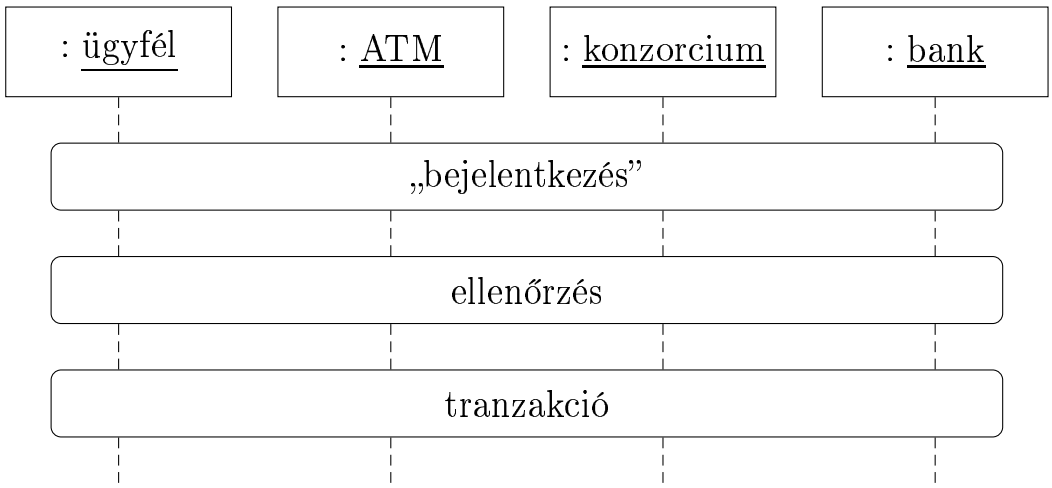

: ügyfél
: ATM
: konzorcium
: bank
kártya be
jelszó?
jelszó
ellenőrzés
ellenőrzés
kérés típusa?
rendben
rendben
típus
összeg?
összeg
tranzakció
tranzakció

2.2. A szekvenciadiagram kiegészítései

A hierarchia ábrázolása

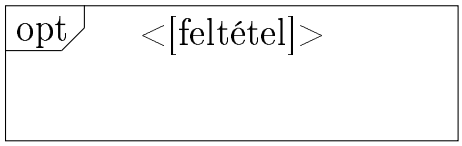
- *Példány dekompozíció:* a szekvenciadiagram egyik objektumát illetve osztályszerepét egy másik szekvenciadiagramban fejtjük ki részletesen. A részletezés során az objektum vagy szerep helyét több objektum vagy szerep veszi át, és pontosítjuk, hogy melyik üzenetet melyik részobjektum küldi illetve fogadja. Lényeges megszorítás, hogy az üzenetek sorrendje a dekompozíció során nem változhat meg.
- *Hivatkozás:* a szekvenciadiagram egy időintervallumnak megfelelő részét kiemeljük egy külön diagramba. Az időintervallumnak megfelelő rész rendszerint egy tevékenységnek felel meg. A hivatkozást egy lekerekített sarkú téglalappal jelöljük, ami egy névvel (azonosítóval) rendelkezik. A kifejtést tartalmazó szekvenciadiagramot ezzel a névvel látjuk el.


: kártyaolvasó
: billentyűzet
: kijelző
kártya be
jelszó?
jelszó
kérés típusa?
típus
összeg?
összeg

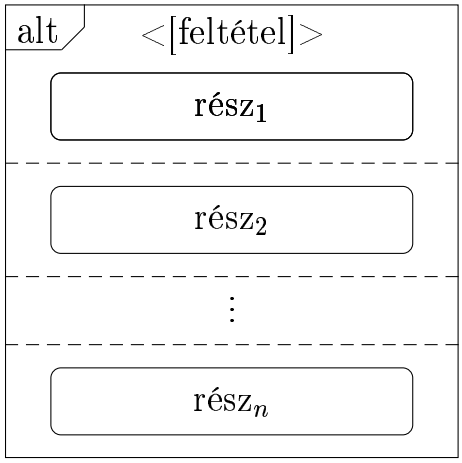

: ügyfél
: ATM
: konzorcium
: bank
„bejelentkezés”
ellenőrzés
tranzakció

A választások ábrázolása

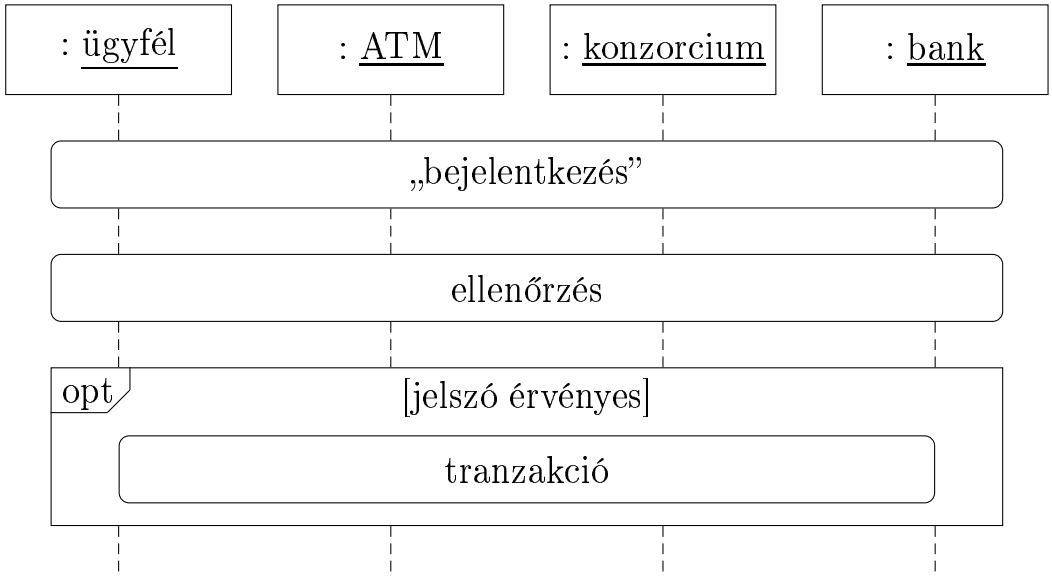
Opcionális rész: egy szekvenciadiagramon belül lehetséges, hogy bizonyos rész üzeneteire csak akkor kerül sor, ha valamilyen feltétel teljesül.



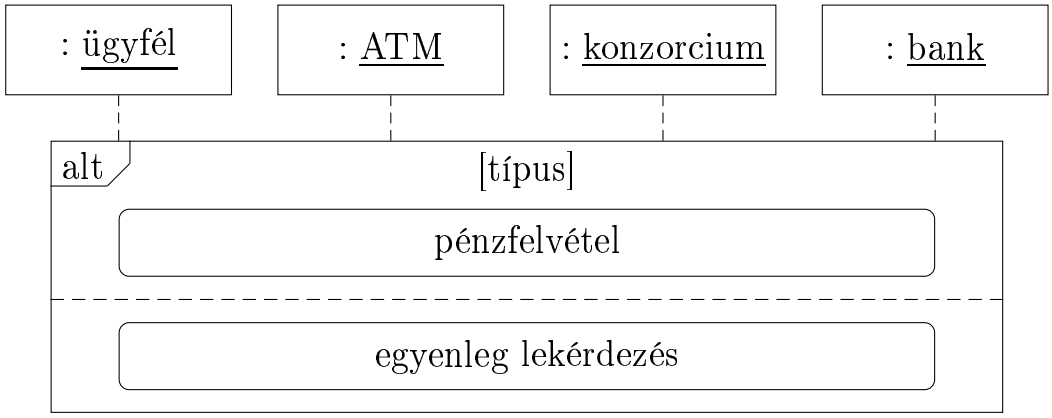
Alternatív rész: előfordulhat, hogy egy kifejezés értékétől függően eltérő részek lehetnek a szekvenciadiagramban.



Például az ATM esetében csak akkor kell tranzakciót végrehajtani, ha az ellenőrzés után tudjuk, hogy a jelszó érvényes:

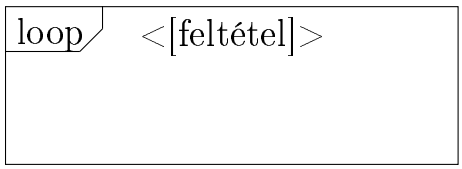


Az ATM esetében mást kell tennünk akkor, ha a kiválasztott művelet a pénzfelvétel illetve az egyenleg lekérdezése.

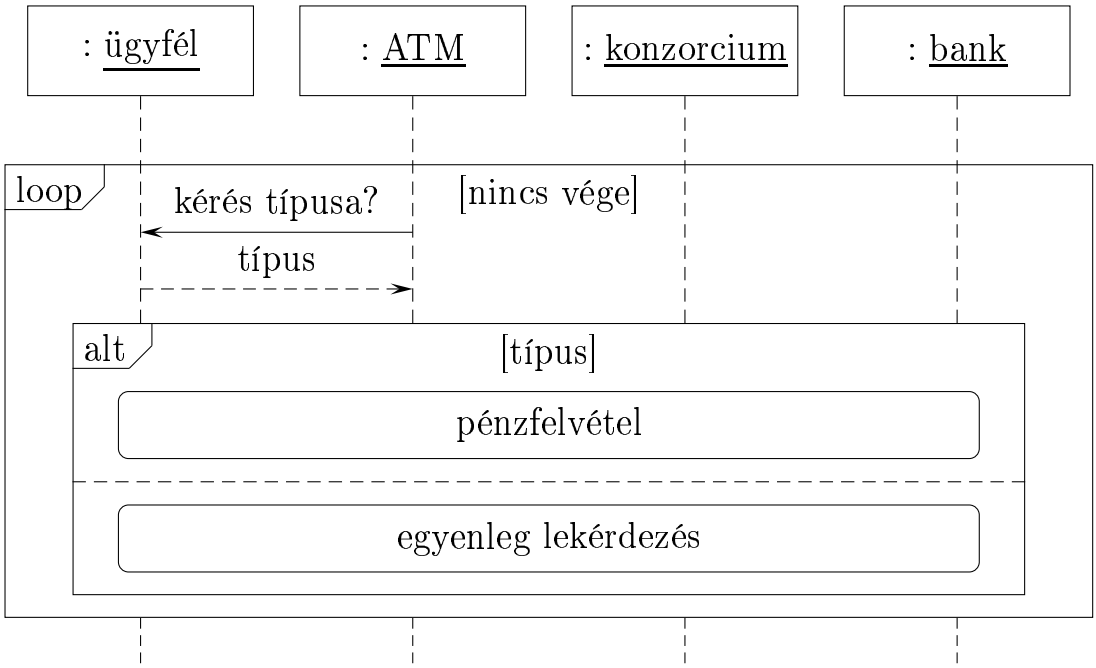


Az iteráció ábrázolása

Egy rendszer működése során bizonyos tevékenységek ciklikusan ismétlődhetnek. Ennek megfelelően a szekvenciadiagram egyes részeinek üzenetei is többször fordulhatnak elő. Az ilyen részt nevezzük iterációs résznek, ciklusnak.

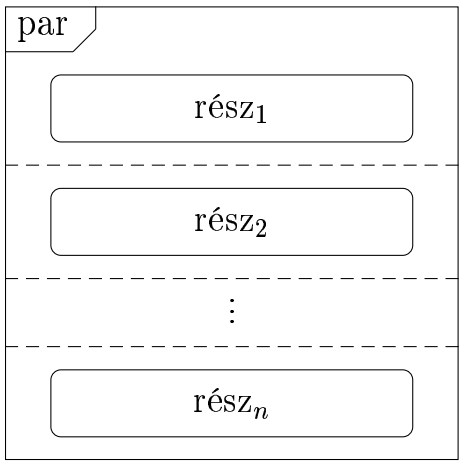


Ha az ATM készülék példájában figyelembe vesszük, hogy több műveletet is végre lehet hajtani egy sikeres bejelentkezés után, akkor a tranzakciós rész egy ciklus lesz, amelyben szerepel az előzőekben megismert alternatív rész.



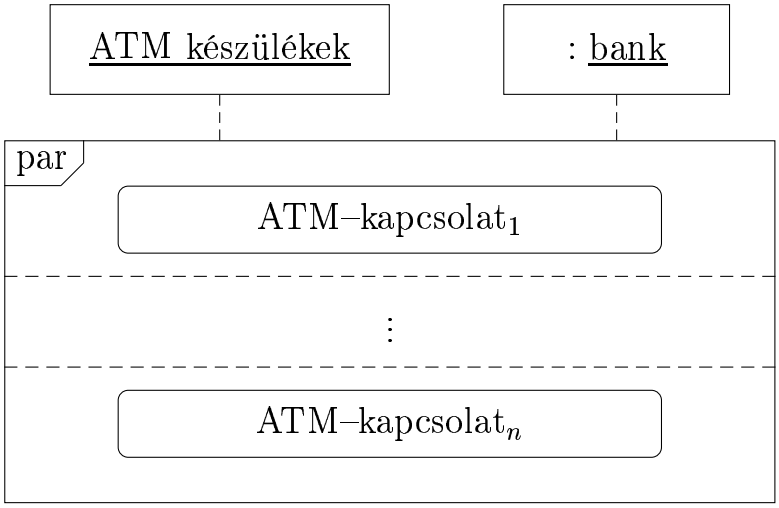
A párhuzamosság ábrázolása

Párhuzamos, osztott rendszerek esetén bizonyos folyamatok egyidőben zajlanak. Ezekben az esetekben a szekvenciadiagramban is biztosítanunk kell a párhuzamosság megjelenését.



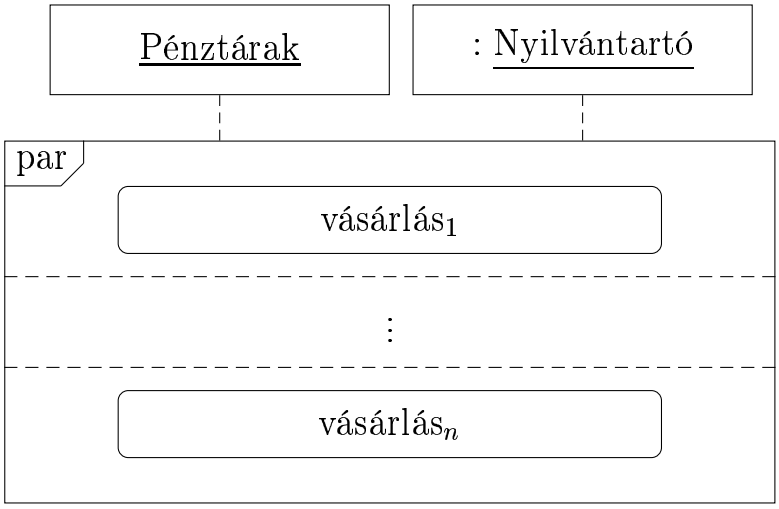
Az egyes részeken belül az üzenetek sorrendje rögzített, csak a különböző részekben szereplő üzenetek időbeliségére nem teszünk megkötést.

Az ATM-mel kapcsolatos példánál maradva, a gyakorlatot figyelembe véve, egy bankban egyidejűleg több aktív ATM-kapcsolat lehet, és mindegyik kapcsolat tevékenységei párhuzamosan folynak. Ha a szekvenciadiagramban az ATM készülékeket összevontan kezeljük, és ezen kívül csak a bankot vesszük figyelembe, akkor a következő az eredmény.

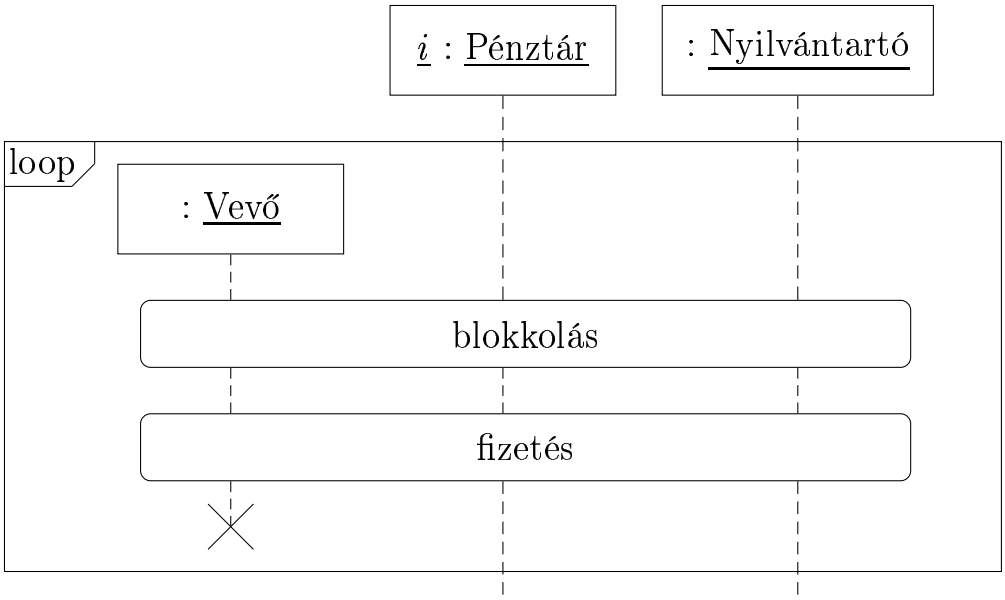


Készítsünk szekvenciadiagramot, amely a következő áruházi vásárlást modellezi! Az áruházban több pénztárnál fizethetnek a vevők, de egy vevő csak egy pénztárnál. A pénztárak párhuzamosan működnek. Egy vevő a kiválasztott pénztárnál felrakja az árukat a szalagra, ahonnan a pénztáros elveszi. A vonalkód alapján a központi nyilvántartásból megállapítja az áru nevét és árát, amit a blokkra nyomtat. Ha végzett a vevő összes árujával, akkor összegzi az árakat. Ezután a vevő fizet. A fizetés történhet készpénzzel vagy kártyával. Készpénzes fizetés esetén a vevő átadja a pénzt, és ha ez meghaladja a blokkon szereplő összeget a pénztáros visszaad. Kártyás fizetés esetén a vevő átadja a kártyát, amit a pénztáros „lehúz", azaz a banknak elküldi a kártyaszámot. Ezután megadja az összeget, amit a bank és a vevő is ellenőriz. Ha rendben van, akkor a vevő megadja a PIN–kódot, amit a bank ellenőriz, és az ellenőrzés eredményét továbbítja a pénztárosnak. Ha a kód rendben van, a pénztáros elismervényt nyomtat, amit átad vevőnek. Ezt a vevő aláírva visszaadja. Ezután a pénztáros átadja az elismervény másolatát. Mindkét fizetési mód esetén a pénztáros a blokkot is átadja. Ezután a pénztáros a következő vevővel foglalkozik.

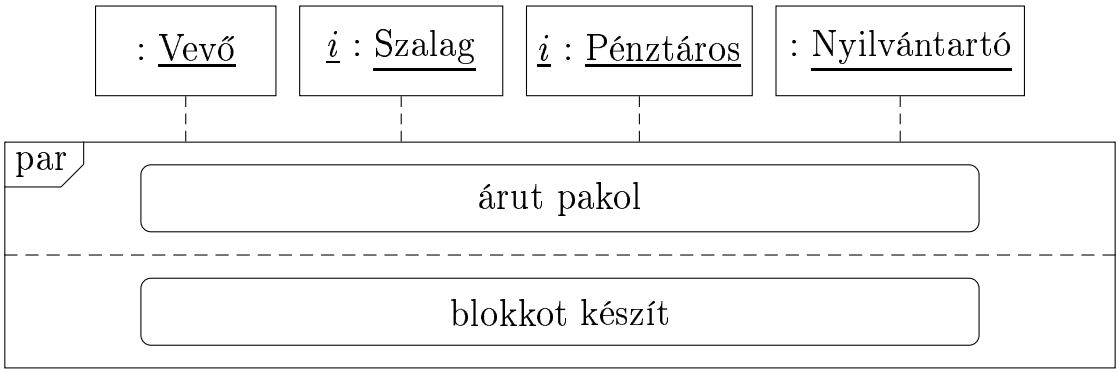
Megoldás:
Ha a pénztárakat összevonva kezeljük, akkor két szerepet különíthetünk el: a nyilvántartót és a pénztárakat. Ha a működő pénztárak száma n, akkor n vásárlási tevékenység zajlik párhuzamosan.



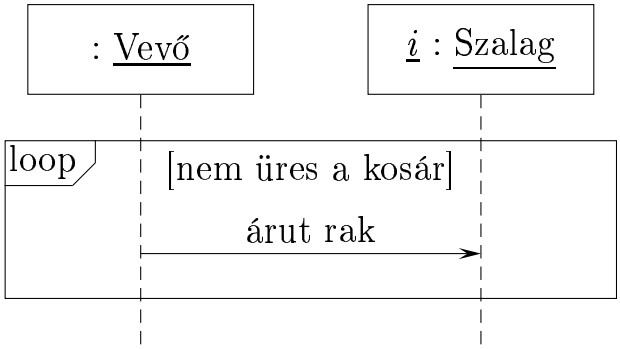
Ezt a diagramot kell finomítani, amelynek során példány dekompozícióval meg kell adnunk a pénztárakat részletesen, másrészt az egyes vásárlási tevékenységeket kell pontosítani. Az i. vásárlás pontosítása során a pénztárakat fel kell bontani az i. pénztárra és az ott található vevőkre.



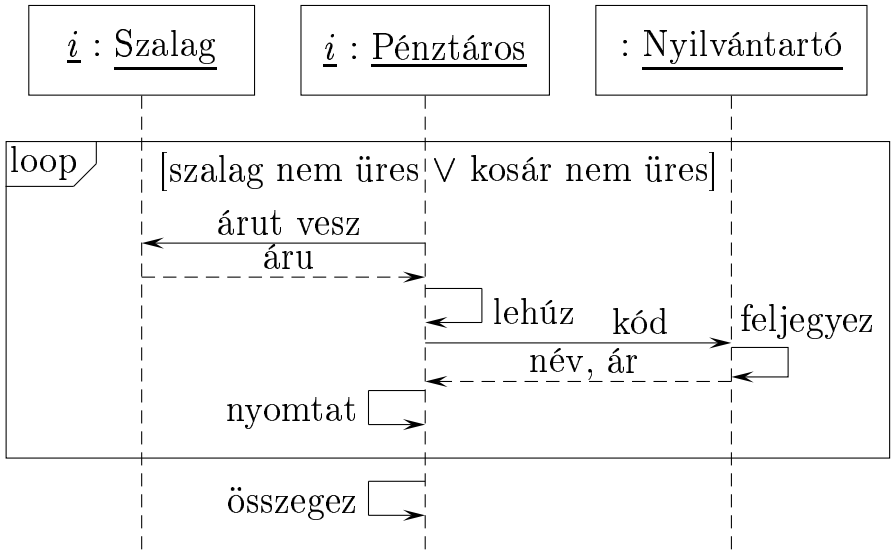
A blokkolás során a pénztárt tovább bontjuk egy pénztárosra és egy szalagra. A tevékenység során a vevő és a pénztáros párhuzamosan dolgozhat. A vevő a szalagra pakol, a pénztáros a szalagról veszi az árukat és blokkol



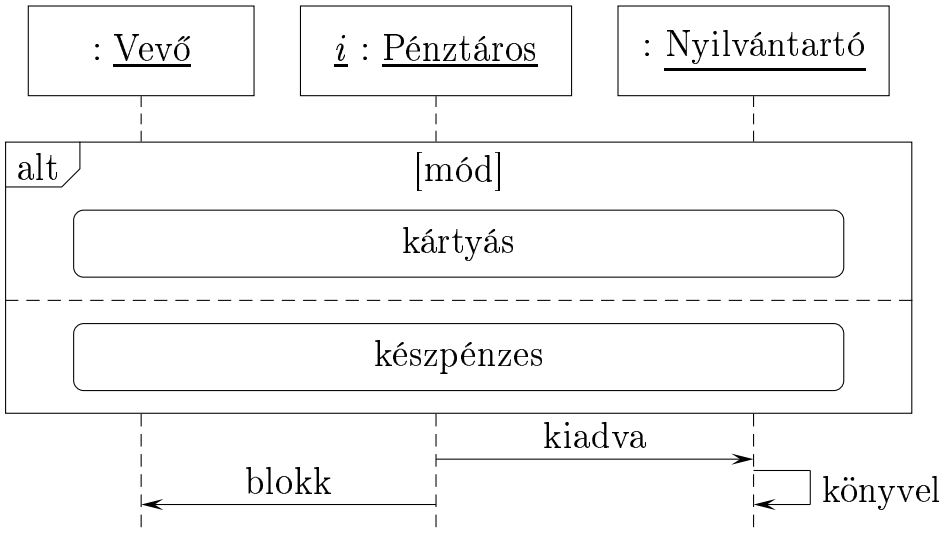
Az áruk pakolása egy ciklus, amelyben a vevő árut rak a szalagra, amíg ki nem ürül a kosara.



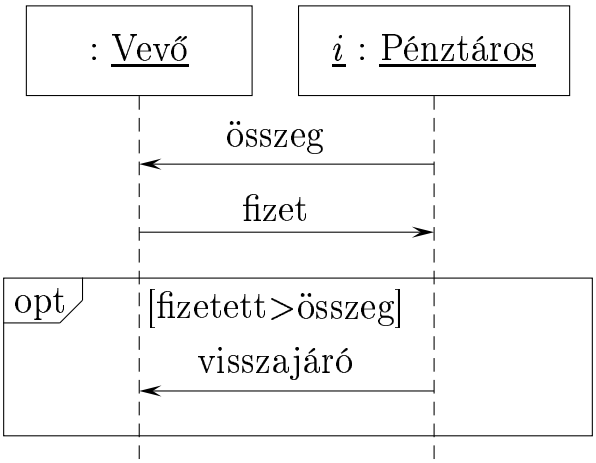
A pénztáros amikor blokkot készít, egy ciklusban árukat vesz a szalagról. Az áru vonalkódját leolvassa, és a kódot a nyilvántartóhoz továbbítja. Innen megkapja a nevet és az árat, amit kinyomtat a blokkra. Közben a nyilvántartó feljegyzi az árucikk kódját, hogy a vásárlás végén csökkenteni tudja a készletet. Az iteráció addig tart, amíg a pénztáros a vevő összes áruját fel nem dolgozza, azaz amíg a kosár nem üres vagy a szalag nem üres. Ezután egy összegzést készít.



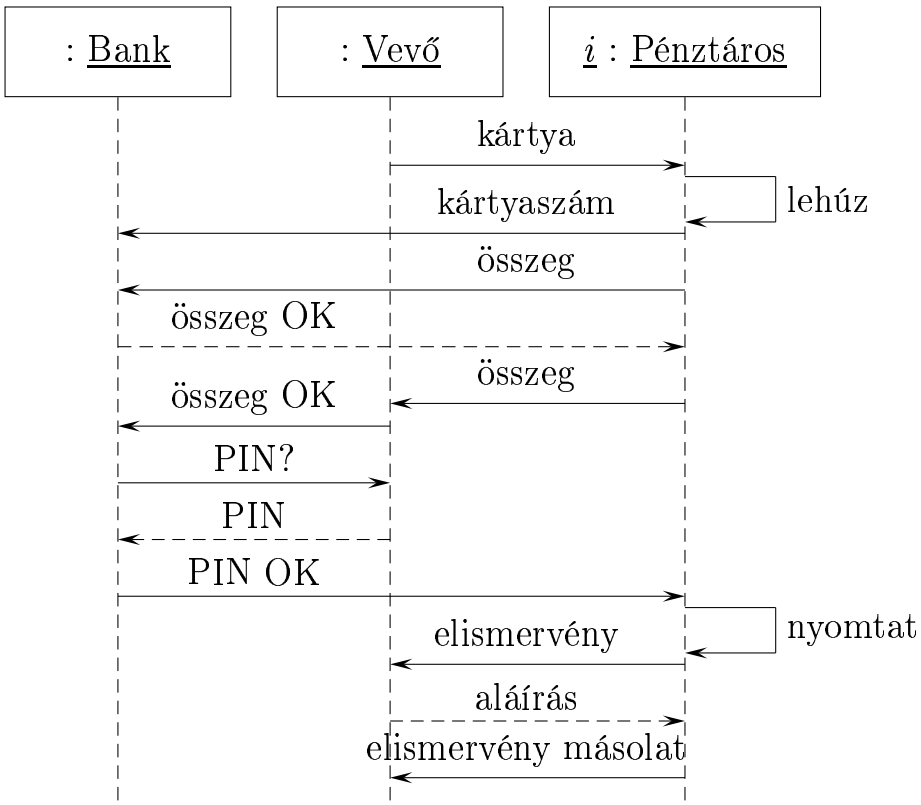
A fizetés lehet kártyás vagy készpénzes. Mindkét esetben az összeg átvétele után a pénztáros üzenet küld a nyilvántartónak, hogy az áruk tényleg ki lettek adva, majd a vevőnek átadja a blokkot. A nyilvántartó a feljegyzések alapján módosítja a készletet.



Készpénzes fizetés esetén a pénztáros megadja az összeget, ezután a vevő fizet. Ha a kifizetett összeg nagyobb, mint a blokkon szereplő összeg, akkor a visszajárót átadja a pénztáros.



Kártyás fizetés során a vevő átadja a kártyát. Ezt a pénztáros lehúzza és a kártyaszámot elküldi a banknak. Ezután az összeget is, amelyről a bank eldönti, hogy kifizethető-e. Ezután a pénztáros megadja a vevőnek az összeget, amit az ellenőriz, és egy gomb megnyomásával nyugtáz a bank felé. Ezután a vevő megadja a banknak a PIN–kódot, amit az ellenőriz, és erről értesíti a pénztárost. A pénztáros kinyomtatja az elismervényt és átadja azt a vevőnek. A vevő ezt aláírva visszaadja. Végül a pénztáros átadja az elismervény másolatát.


: Bank
: Vevő
i : Pénztáros
kártya
kártyaszám
lehúz
összeg
összeg OK
összeg
összeg OK
PIN?
PIN
PIN OK
elismervény
nyomtat
aláírás
elismervény másolat

Feladat: végiggondolni, hogy az esetleges hibák, rendellenességek kezelésével miként lehet a diagramokat kiegészíteni. Néhány lehetséges eset, amelyeket egy valós rendszerben figyelembe kell venni.

- Az áru lehúzásakor nem sikerül a vonalkódot módosítani.
- Nincs elég készpénz a vevőnél, és árukat kell „visszavenni”, amíg az összeg már megfelelő.
- Kártyás fizetés esetén nem sikerül a kártyát leolvasni, és néhányszor újra kell próbálkozni.
- Az összeg túllépi a kártyával fizethető maximális összeget, és vissza kell venni árukat, vagy nem lehet kártyával fizetni.
- Nem jó a megadott PIN–kód, és újra kell próbálkozni.
- Ha nem lehet kártyával fizetni, akkor meg kell próbálni a készpénzes fizetést. Ha arra nincs lehetőség, az egész vásárlást törölni kell.

2.3. Végrehajtási gráf

A végrehajtási gráf egy irányított gráf, amelynek csúcsai a rendszer tevékenységeit, élei pedig a tevékenységek közötti sorrendet ábrázolják. A rendszer tevékenysége egy feldolgozási lépés, amely valamilyen feladatot hajt végre a rendszerben. Ez lehet művelethívások gyűjteménye illetve egyszerű utasítás is.

A gráf csúcsait megkülönböztetjük annak alapján, hogy milyen feldolgozási lépésnek feleltethetőek meg.

Egyszerű csúcs: egy funkcionális komponens, amely az adott szinten tovább nem bontható.

Kiterjesztett csúcs: olyan feldolgozási lépés, amelyet egy másik végrehajtási gráfban fejtünk ki részletesen.

Ismétlési csúcs: az ezt követő csúcsok n-szer ismétlődnek; a ciklus utolsó csúcsából ebbe a csúcsba mutat él.

Választási csúcs: az ehhez csatolt csúcsok végrehajtása feltételes; minden esethez tartozik egy végrehajtási valószínűség.

Párhuzamossági csúcs: a csatolt csúcsok tevékenységei párhuzamosan hajtódnak végre, és mindegyiknek be kell fejeződnie, mielőtt ebből a csúcsból tovább lépnénk.

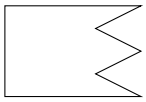
Hasító csúcs: a csatolt csúcsok párhuzamosan futó új szálakat reprezentálnak, amelyek befejeződése előtt is tovább mehetünk.

Egyszerű csúcs

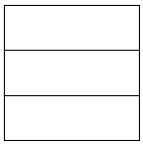
Kiterjesztett csúcs

Ismétlési csúcs

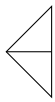
Választási csúcs



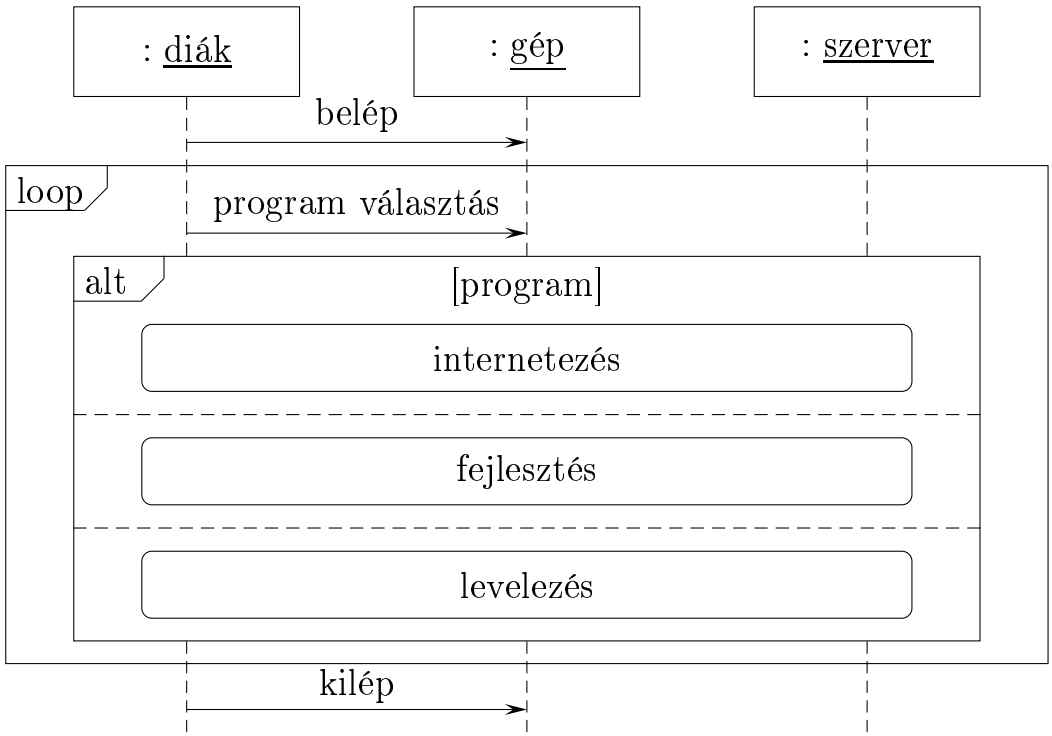
Párhuzamossági csúcs



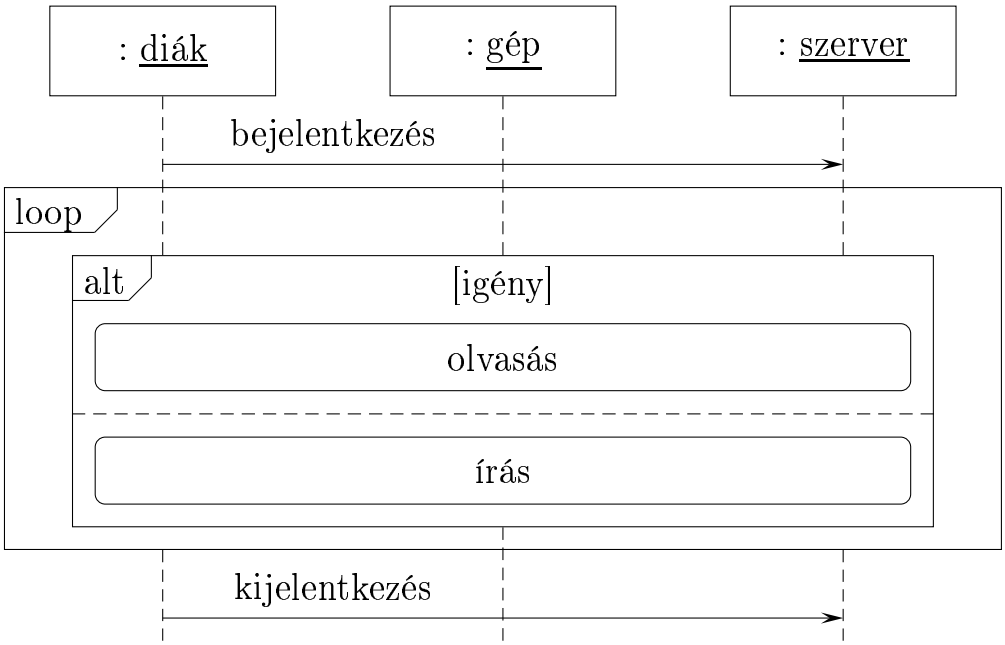
Hasító csúcs



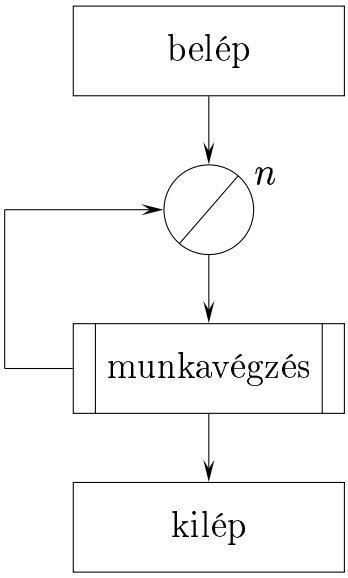
A hallgatók egy gépteremben levő számítógépeket használnak feladataik elvégzéséhez. Egy hallgató először belép a gép rendszerébe, majd a feladatait végzi el, végül kilép a rendszerből. Háromféle feladata lehet egy hallgatónak: internet használat, fejlesztés és levelezés. Minden esetben kiválasztja a megfelelő programot a feladathoz. Az internetezést és a fejlesztést ezen a szinten nem bontjuk tovább. A levelezés során előbb bejelentkezik a szerverre, üzeneteket olvas vagy ír, a végén pedig kijelentkezik.

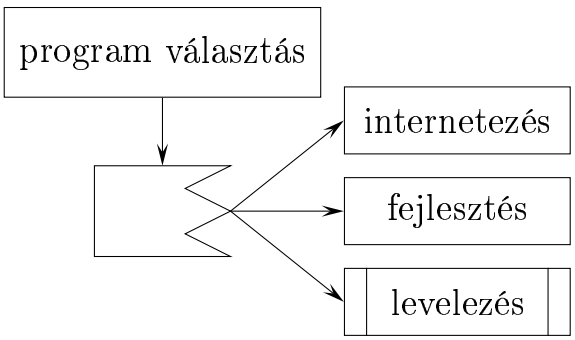

: diák
: gép
: szerver
belép
loop
program választás
alt
[program]
internetezés
fejlesztés
levelezés
kilép

Levelezés:

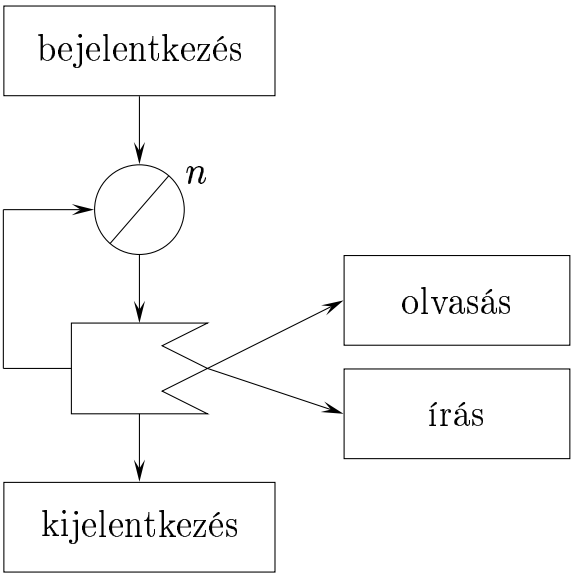

: diák
: gép
: szerver
bejelentkezés
loop
alt
[igény]
olvasás
írás
kijelentkezés

A szekvenciadiagram alapján adódik a végrehajtási gráf, hiszen egy diagramban előforduló iterációnak egy ismétlési csúcs és a ciklusmagot leíró kiterjesztett csúcs felel meg, az alternatív szerkezetnek pedig egy választási csúcs.





Levelezés:



Párhuzamos rendszerek esetén az eddig megismert csúcsokat ki kell egészítenünk szinkronizációs csúcsokkal, amelyek két csoportra oszthatók. Az egyik csoport csúcsai a hívó folyamatra vonatkoznak, a másik típusú csúcsok a hívott folyamathoz kapcsolhatók.

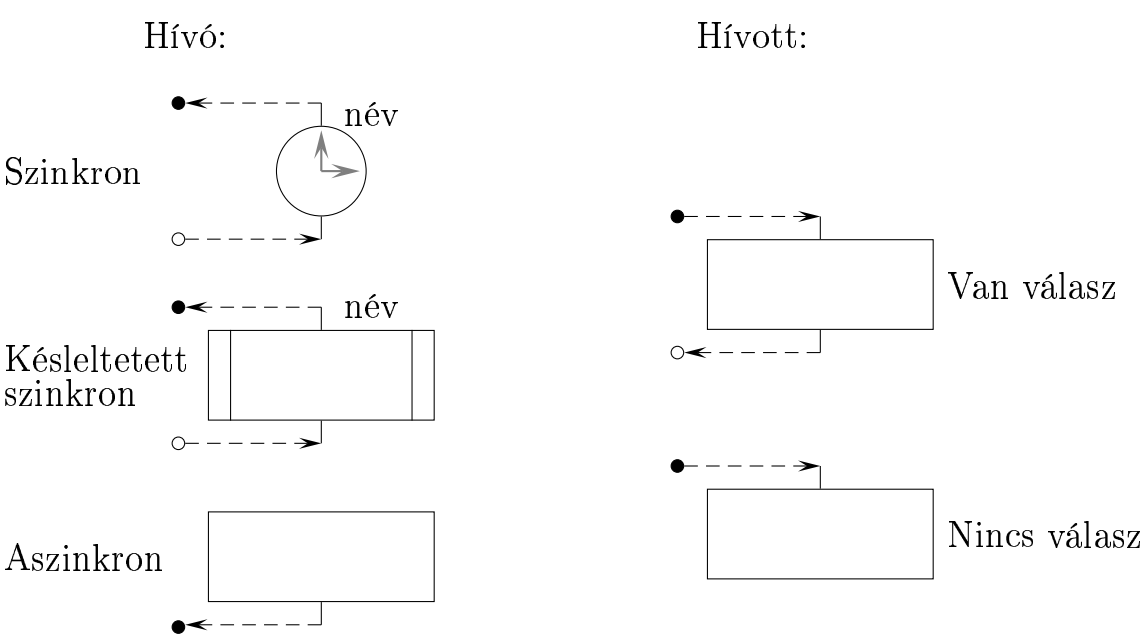
A hívó folyamatra vonatkozó csúcsok:

Szinkron hívás: a hívó a válaszra vár.

Késleltetett szinkron hívás: a hívó a válaszra vár, közben azonban a feldolgozás zajlik.

Aszinkron hívás: nincs válasz, nem kell várni.

A hívott folyamatra vonatkozó csúcsoknál csak azt kell jelölnünk, hogy van-e válasz.

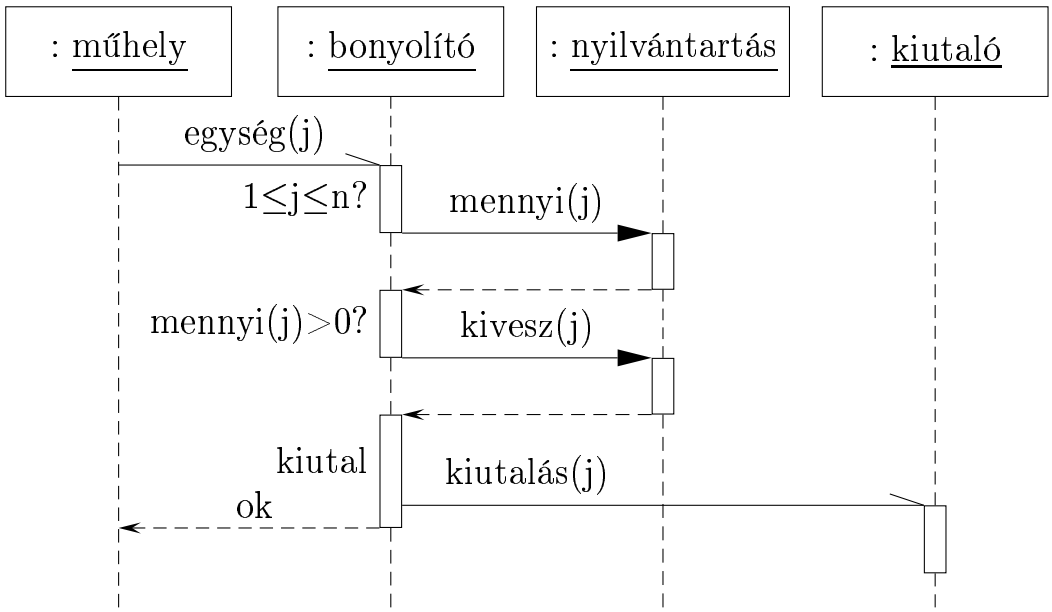

Hívó:
Hívott:
név
Szinkron
Van válasz
név
Késleltetett
szinkron
Nincs válasz
Aszinkron

Egy üzemben a műhelyt egy raktárból látják el a termeléshez szükséges anyagokkal, árucikkekkel. A raktárban n különböző árucikket tárolnak, amelyeket az $\{1, \ldots, n\}$ értékekkel azonosítanak. Minden cikkről nyilvántartják a raktáron található mennyiséget. A raktári nyilvántartáson három művelet hajtható végre.

- Egy darab j azonosítójú áru elhelyezése a raktárba: *betesz(j)*.
- Egy darab j azonosítójú áru kivétele a raktárból: *kivesz(j)*.
- A j azonosítójú áru raktári készlete: *mennyi(j)*.

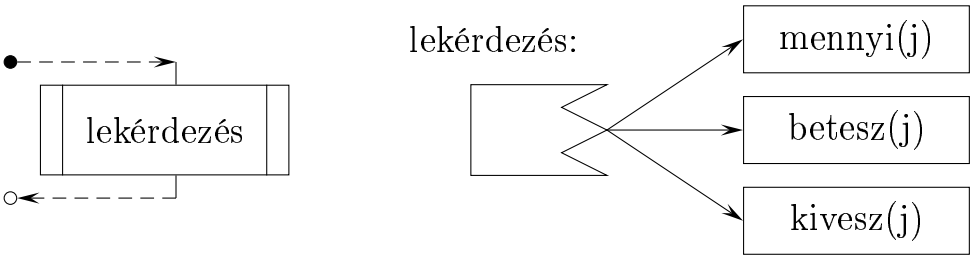
(Az eddigiek alapján látható, hogy a raktári nyilvántartás tulajdonképpen egy zsák.)

A nyilvántartás egy központi számítógépen helyezkedik el. Ugyanezen a központi gépen található a kiutalást kezelő program (kiutaló) is. A j azonosítójú árucikk rendelését a műhely egy terminálon (PC) keresztül kezdeményezi az *egység(j)* kéréssel, amit a kéréseket lebonyolító programnak (bonyolító) továbbít. Ez egy hálózati szerveren található, és ellenőrzi a kérést. Ennek során megnézi, hogy olyan egységet kérnek-e, ami a raktárkészlet profiljába tartozik ($1 \leq j \leq n$), és ha igen, akkor van-e raktáron a kért egységből ($mennyi(j) > 0$). A teljesíthető rendelést a bonyolító kezeli, azaz kezdeményezi a készlet megváltoztatását, és a kiutalást; a rendelés teljesítéséről értesíti a műhelyt.

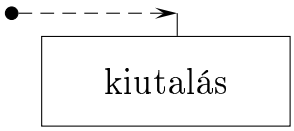

: műhely
: bonyolító
: nyilvántartás
: kiutaló
egység(j)
1≤j≤n?
mennyi(j)
mennyi(j)>0?
kivesz(j)
kiutal
kiutalás(j)
ok

A végrehajtási gráfot három objektum esetében kell megadnunk. Ezek a bonyolító, a nyilvántartás és a kiutaló.

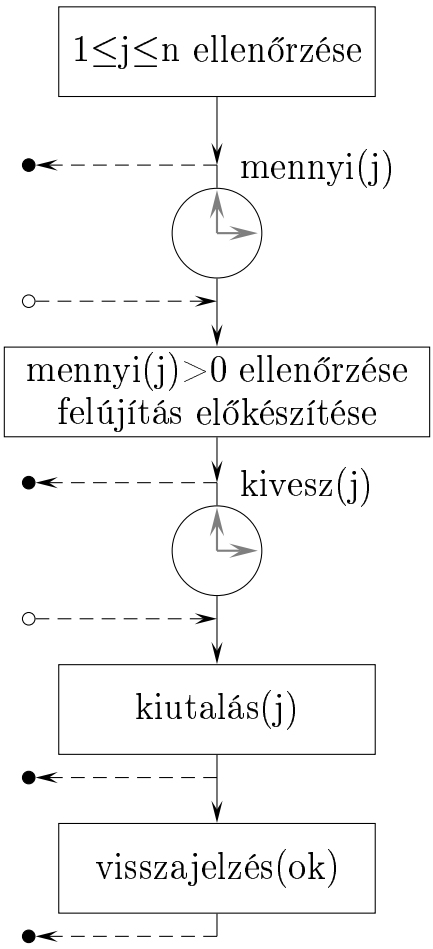
A nyilvántartás minden művelete, nevezzük ezeket együttesen lekérdezésnek, egy hívott objektum választ adó művelete lesz. A lekérdezésen belül három művelet lehet, amelyeket nem bontunk tovább.



A kiutalónak egy válasz nélküli művelete van, a kiutalás.

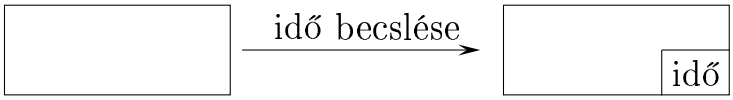


A bonyolító esetében az első tevékenység egy ellenőrzés, amit a nyilvántartás mennyi műveletének szinkron hívása követ. Ezután az eredményt ellenőrizni kell, és a felújítást elő kell készíteni. Ezt követi a nyilvántartás kivesz műveletének szinkron hívása. Végül a kiutalás és a visszajelzés aszinkron végrehajtása következik.

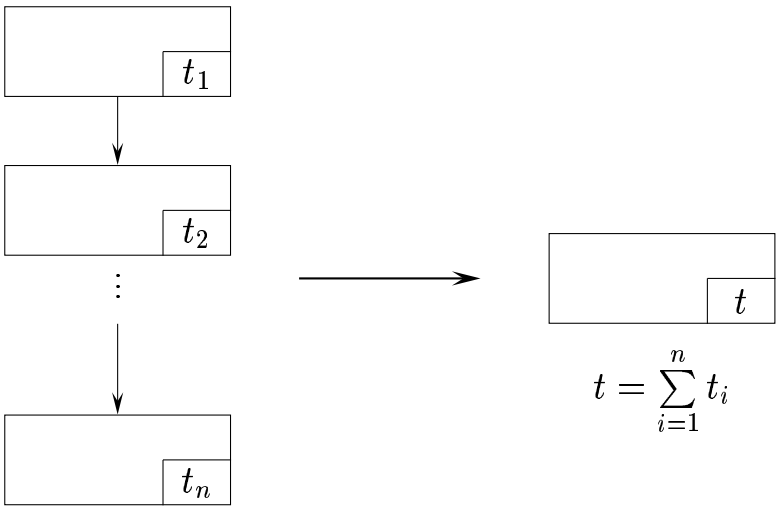

1≤j≤n ellenőrzése
mennyi(j)
mennyi(j)>0 ellenőrzése
felújítás előkészítése
kivesz(j)
kiutalás(j)
visszajelzés(ok)

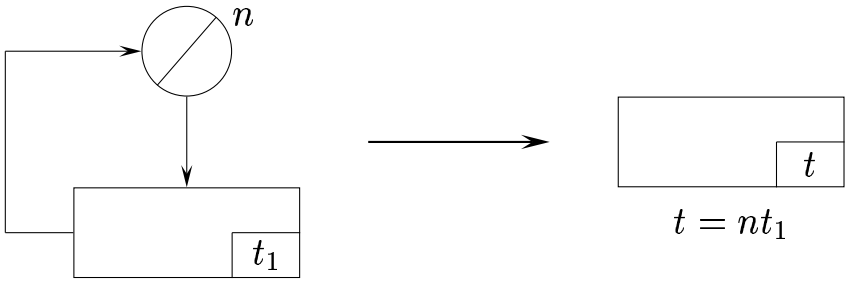
2.4. A futási idő becslése

Az idők becslése a végrehajtási gráf egyszerűsítésén alapul. Tegyük fel, hogy minden egyszerű csúcsban szereplő művelet végrehajtási idejét meg tudjuk becsülni. Ez nem feltétlen pontos szám, lehet alsó korlát, felső korlát és átlagos eset is. Ezeket a csúcsokat „számított" csúcsokra cseréljük.

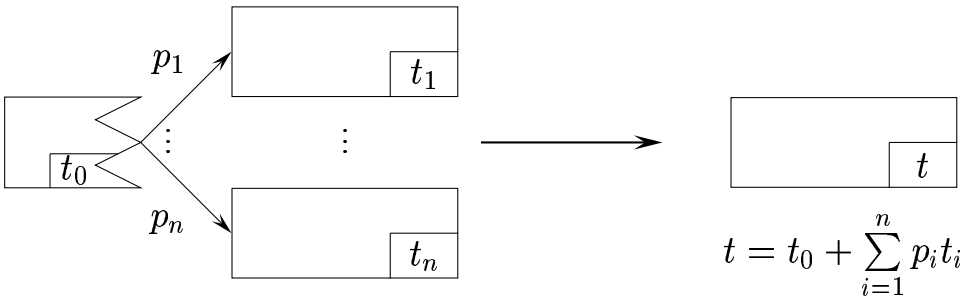


A végrehajtási gráf alapvető szerkezetei a szekvencia, az iteráció és a választás. Szekvencia esetén a benne szereplő számított csúcsokat helyettesíteni lehet egy számított csúccsal, amelynek ideje a szekvenciában szereplő csúcsok idejeinek összege. Iteráció esetén a ciklusmag idejét meg kell szorozni a végrehajtások számával. Mindhárom (optimális, legrosszabb, átlagos) idő meghatározása esetén ezt kell tennünk, csak a megfelelő időt kell figyelembe venni a felhasznált számított csúcsokban.


t_1
t_2
t_n
t
$t = \sum_{i=1}^{n} t_i$


n
t_1
t
$t = nt_1$

Választás esetén figyelembe kell vennünk a feltétel kiértékelési idejét, amit növelni kell a legrövidebb idejű esettel az optimális, a leghosszabb idejű esettel a legrosszabb idő meghatározásához. Az átlagos idő számításakor a kiértékelési időhöz az egyes ágak valószínűségekkel súlyozott idejét kell adnunk.



Az egyszerűsítéseket addig kell folytatni, amíg egyetlen csúcs marad. A csúcs ideje tartalmazza a becsült végrehajtási időt. Az egyszerűsítéseket célszerű automatizálni, léteznek erre megfelelő eszközök. Itt most csak szekvenciális rendszerekre illusztráltuk a módszert. Párhuzamos rendszerekben a párhuzamossági és hasító csúcsokat is kezelni kell, és a szinkronizációt is figyelembe kell venni.

2.5. Végrehajtási rendszermodell

A végrehajtási gráf csúcsaiban szereplő tevékenységek futási idejének meghatározásakor figyelembe kell venni, hogy a folyamatok nem önmagukban léteznek. Egyidőben több folyamat futhat, és ezek közös erőforrásokat használhatnak. Az erőforrásokra esetleg várakozni kell, ami növelheti a futási időt. Ezt az eddigiekben nem vettük figyelembe. Ha ez befolyásolja a rendszer viselkedését, akkor ezt is tartalmaznia kell a számításoknak.

Több folyamat lehet, mert:

- eleve több felhasználóból, folyamatból áll a rendszer (egyszerre számos ATM tranzakció történhet egy bankban);
- más rendszerek is ugyanazokat az erőforrásokat használják (a bankban az ATM-ek által használt számítógépen egyéb programok – nyilvántartás, jelentés készítés, átutalások – futhatnak);
- az alkalmazás több processzből áll.

Egy gráffal szemléltetjük ezt a modellt is. A modell alapfogalmai, illetve a gráf csúcsai:

Szerver: egy komponens, amely valamilyen szolgáltatást nyújt a szoftvernek (processzor, lemez, hálózati elem). Rendszerint egy sor tartozik hozzá.

Sor: ebben várakoznak a kiszolgálandó feladatok. Ha a szerver foglalt egy feladat érkezésekor, a feladat ebbe a sorba kerül, és itt vár, amíg a szerver szabad nem lesz.

Forrás: a tevékenység kezdete, eredete.

Nyelő: a tevékenység befejezése, kilépés.

Késleltetés: aktív erőforrás sor nélkül.

Találkozási pont: több él is bevezethet ebbe a csúcsba, és innen több él is kiindulhat.

Almodell: ez a csúcs egy később részletesen kifejtett modellt reprezentál.

Almodell belépési pont: az almodellel leírt rész tevékenységeinek kezdete.

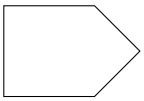
Almodell visszatérési pont: az almodellel leírt rész tevékenységeinek vége.

Sor és szerver

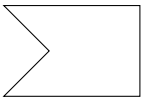
Késleltetés

Almodell

Forrás

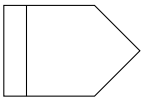


Nyelő

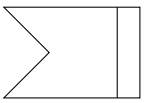


Találkozási pont

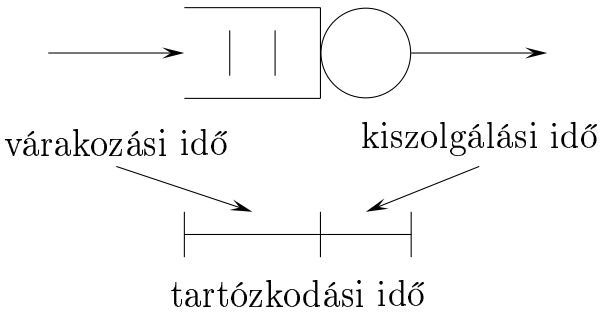
Almodell
belépési pont



Almodell
visszatérési pont

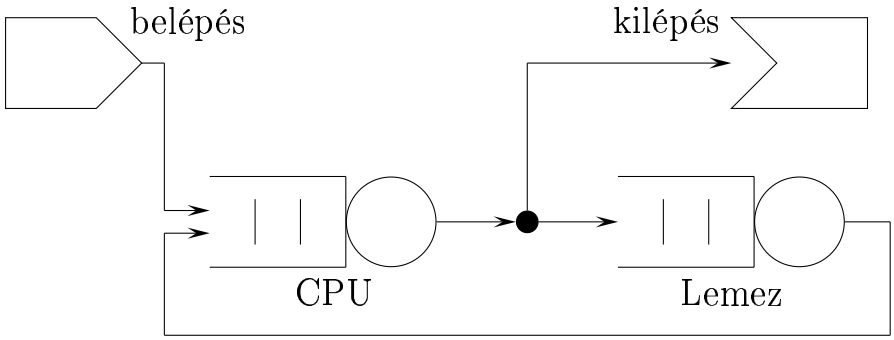


A futási idő meghatározásánál figyelembe kell venni, hogy egy folyamat mennyi ideig tartózkodik egy kiszolgáló egységnél. A tartózkodási idő a várakozási idő és a kiszolgálási idő összege.

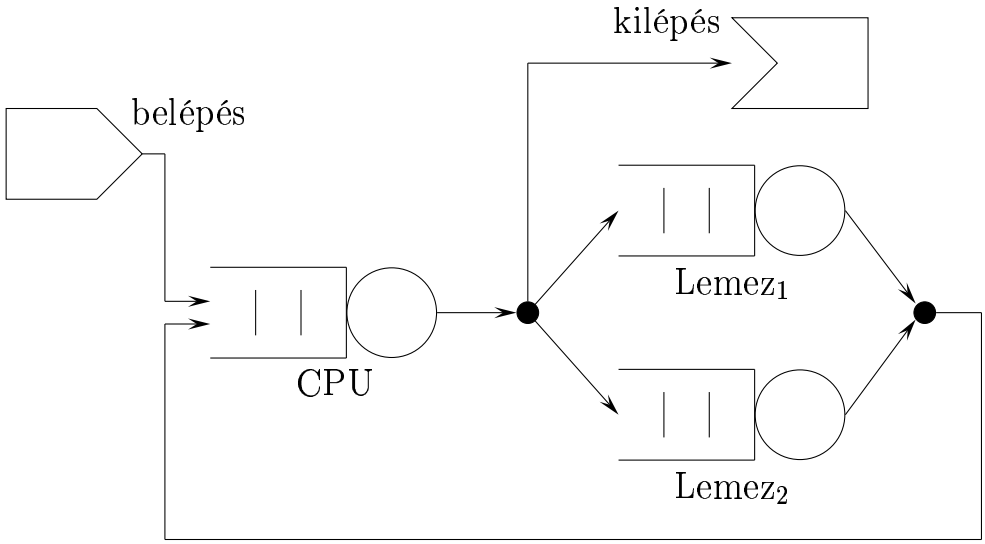


Számunkra nyilvánvalóan a tartózkodási idő az érdekes. Ezeket összegezve kapjuk meg egy tevékenység végrehajtási idejét.

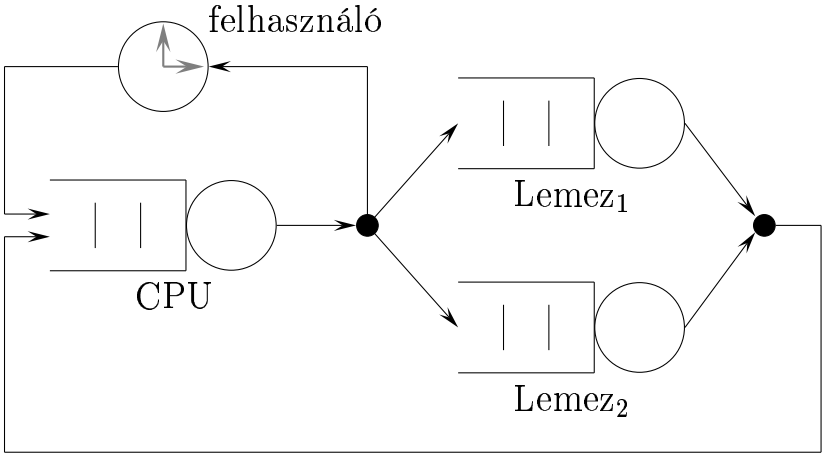
Egy egyszerű végrehajtási rendszermodell a következő.



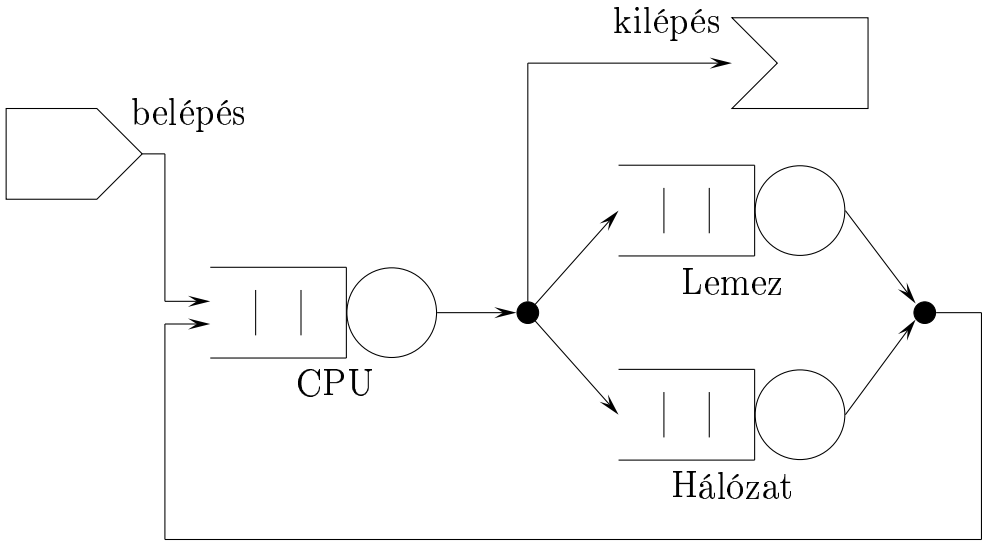
Nyílt modell:



Zárt modell:

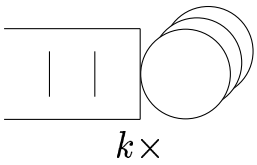


ATM tranzakció végrehajtása a banknál:

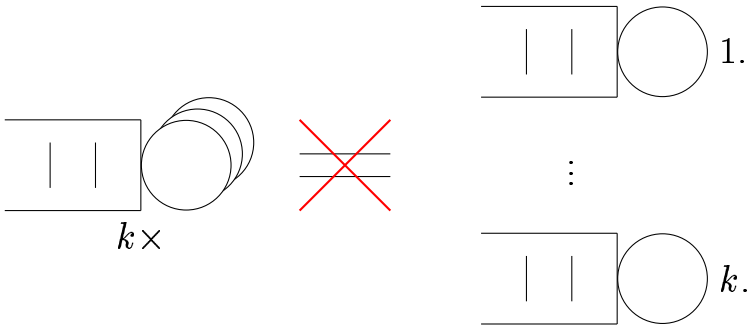


1. Felhasználó azonosítása: CPU, lemez.
2. Tranzakció ellenőrzése: CPU, lemez.
3. Eredmény rögzítése, és közlése: CPU, lemez, hálózat.

Ha több szerverhez tartozik egy sor, akkor annak jelölése:



Ez nem ugyanaz, mint több, sorral rendelkező szerver, mert ott amikor egy folyamat bekerül egy sorba, akkor már eldőlt, hogy melyik szerverhez kerül. (Nem mindegy, hogy például egy postán az összes ablakhoz egy sor tartozik, vagy minden ablaknál van külön-külön sor.)



Egy lehetséges példa egy többprocesszoros gép, ahol a feladatok egy sorba kerülnek, de mindig az éppen szabad processzor hajtja végre a feladatot.