

3. gyakorlat – Folyamatmodellek, kooperáló viselkedésmodellek – Megoldások

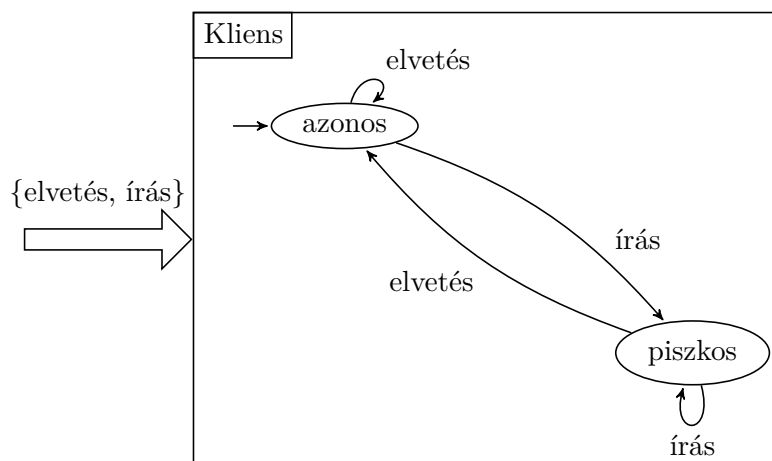
1. Összetett rendszer modellezése

Felhő alapú adattárolást modellezünk (ld. Dropbox, Google Drive, Tresorit), egyetlen állományra szorítkozva. Az állománynak a szerveren és a kliensnél (pl. laptop) is elérhető egy-egy replikája, kezdetben azonos tartalommal. A fájl módosításai *szinkronizálás* során továbbítódnak a példányok között. Ha szinkronizáció előtt mindkét példányt módosítják, akkor *ütközés* lép fel, amelyet a felhasználónak kell feloldania a kliensen.

Lokálisan a kliensnél, illetve (pl. másik kliens tevékenységének hatására) módosulhat a szerveren is. Felhasználói utasításra, valamint időről időre spontán módon a kliens és a szerver szinkronizálhat; ilyenkor az esetleges módosítás eljut a másik példányhoz is, és újra *azonos* lesz a két másolat. Ha a legutóbbi szinkronizáció óta egymástól függetlenül mindkét replikát módosították, akkor viszont konfliktushelyzet (ütközés) áll fenn. Ilyenkor a kliens a saját és a szerverről letöltött változatot összehasonlítja, és a felhasználóra bízva az ütközés feloldását.

- a) Modellezzük először a kliens (részleges) működését állapotgéppel! A kliens kezdetben *azonos* állapotú (a lokális fájlmásolat egyezik azzal, ami a szerveren a legutóbbi szinkronizációkor volt / lett), ám *írás* bemenet hatására a *piszkos* állapotba kerül (és további *írás* hatására is ottmarad). Az *elvetés* bemenet hatására tetszőleges állapotból újra *azonos* állapotba kerül.

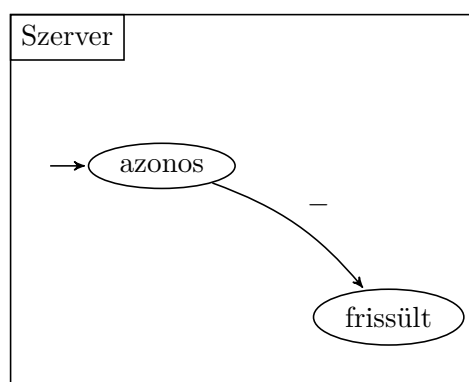
Megoldás



- b) A szerver lehetséges állapotai (csupán az adott klienssel való szinkronizációt vizsgálva) az *azonos* és a *frissült*. Előfordulhat, hogy a megfelelő írási jog birtokában egy másik felhasználó (vagy ugyanazon felhasználó egy másik kliens, pl. a telefonja segítségével) frissíti a szerveren található állományt. A szerver kezdetben *azonos* állapotban van.

Megoldás

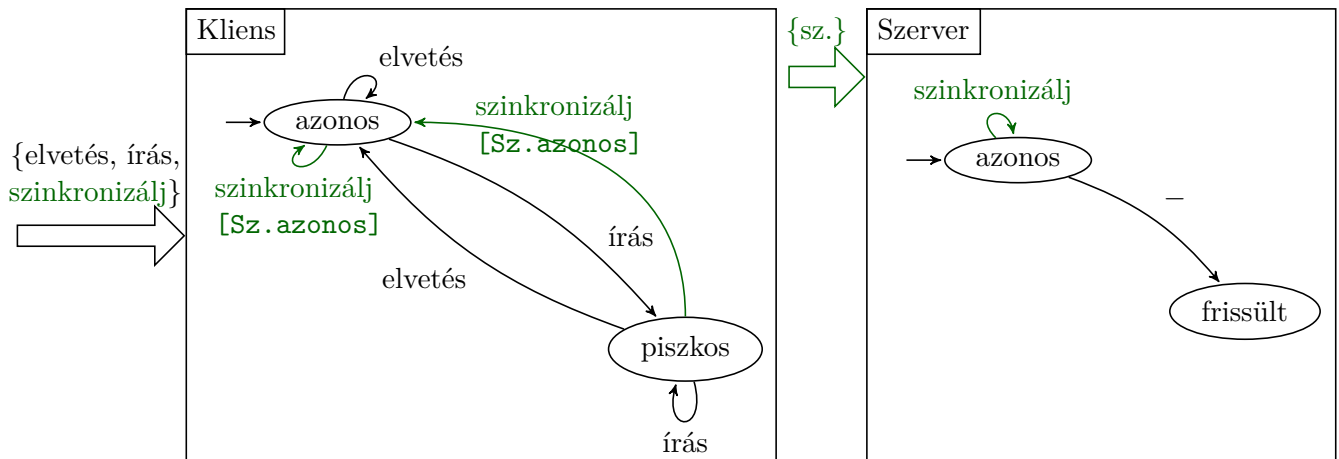
Itt annyi az érdekesség, hogy spontán állapotátmenet lesz (már ha a lokális felhasználtól érkezett bemenetekre figyelünk csak és nem modellezzünk másik klienst, pl. „klienst2”).



- c) Ha a szerver *azonos* állapotban van, akkor a kliens a *szinkronizálj* bemenet hatására feltölti az esetleges lokális módosításokat a szerverre, és szintén *azonos* állapotba kerül. A *szinkronizálj* bemenetet a szerver is megkapja. Hol kooperál a két állapotgép?

Megoldás

A Kliens állapotgép két új átmenetere **[Szerver.azonos]** őrfeltételt rakunk. Érdekesség, hogy ez a másik állapotgép pillanatnyi állapotától függ – ez tehát kooperáció! A modell absztrahálja a valóságot, itt ténylegesen nyilván üzenetcsere van a kliens és a szerver között, ahol kicserélik ezt az információt, ill. a fájlt is fel kell tölteni.



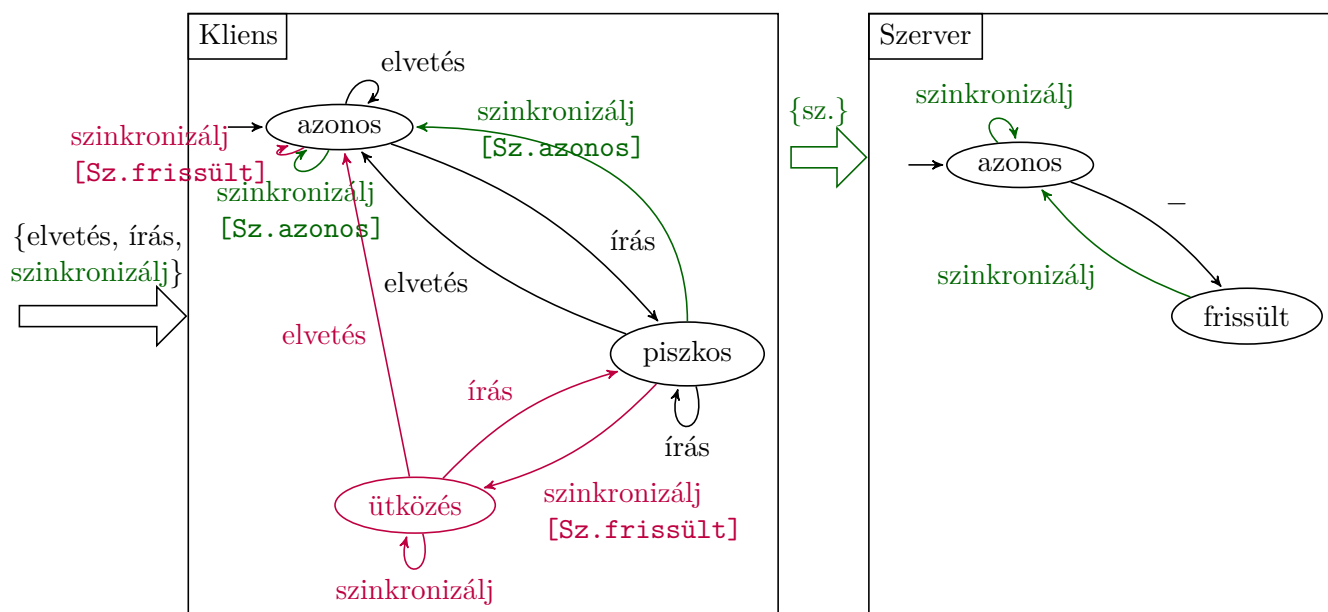
- d) Ha a szerver *frissült* állapotban van, akkor a kliensnek adott *szinkronizálj* bemenet hatására a szerver *azonos* állapotba kerül; a kliens pedig *azonos* állapotból nem mozdul, de *piszkos* állapotból *ütközés* állapotba megy. Mit jelent ez? Mi történjen az *ütközés* állapotban? Hol kooperál a két állapotgép?

Megoldás

Itt egyszerre lép a két állapotgép, tehát a *szinkronizálj* bemenet mindkét automatát befolyásolja! (Ilyenkor azt mondjuk, hogy nem aszinkron, hanem vegyes szorzatban van a két komponens automatája: főleg aszinkron lépnek, de néha szinkron, mert mindkettő egyszerre lép.)

A kliens állapotgép új élei **[Szerver.frissült]** őrfeltételt kapnak. Vegyük észre, hogy a *Kliens.azonos* állapotból két hasonló átmenet megy ki ellentétes őrfeltétellel: (**[Szerver.azonos]**, ill. **[Szerver.frissült]**), amik összevonhatóak egyetlen őrfeltétel nélküli éllé (a lenti ábrán ez nem szerepel). Ez nyilván az az eset, amikor a kliens detektálja az *ütközést*. Ilyenkor a szerver *azonos* állapotba megy, mivel önála a legutolsó szinkronizáció óta nincs újabb. A kliensen kell feloldani a konfliktust. Az *ütközés* állapotban az eseményekre például így léphetünk:

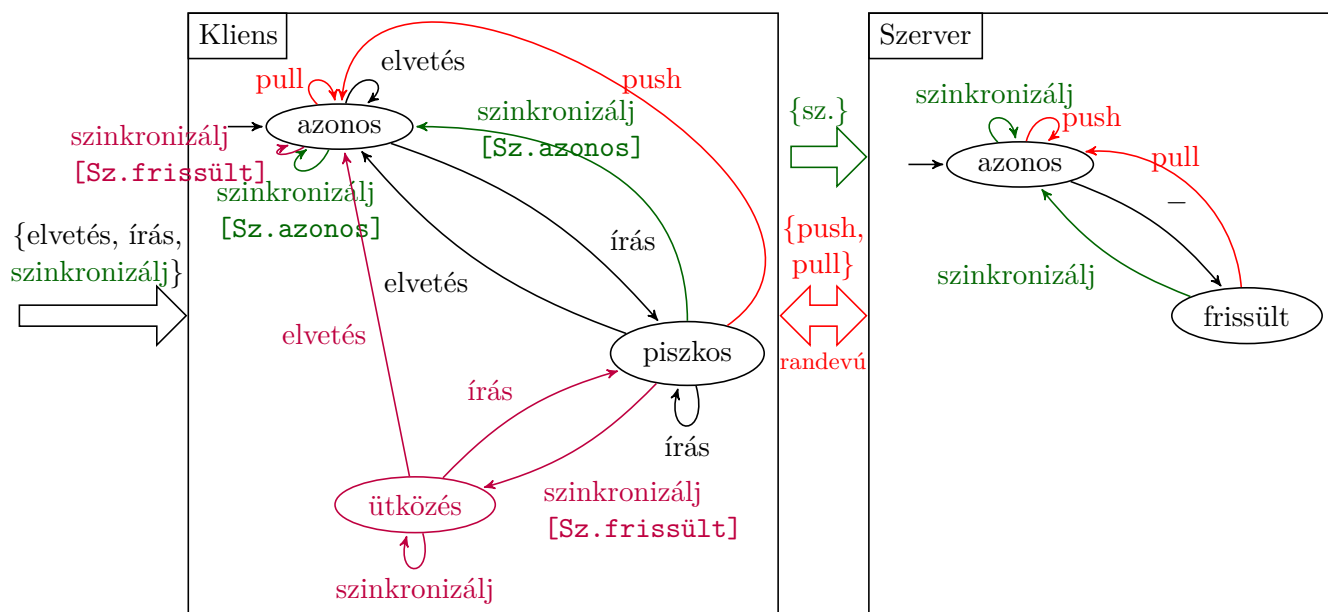
- *elvetés* → *azonos*,
- *írás* → *piszkos*,
- *szinkronizál* → *ütközés*.



- e) A kliens időnként magától is szinkronizál a szerverrel, felhasználói bemenet nélkül. Mit jelent ez? Hol kooperál a két állapotgép?

Megoldás

Ugyanaz, mint előbb, csak külső bemeneti esemény helyett közös, belső randevú esemény (legyen **pull** és **push**) hatására. Opcionális változtatás lehet, pl. konfliktust ilyenkor sose idézzon elő.

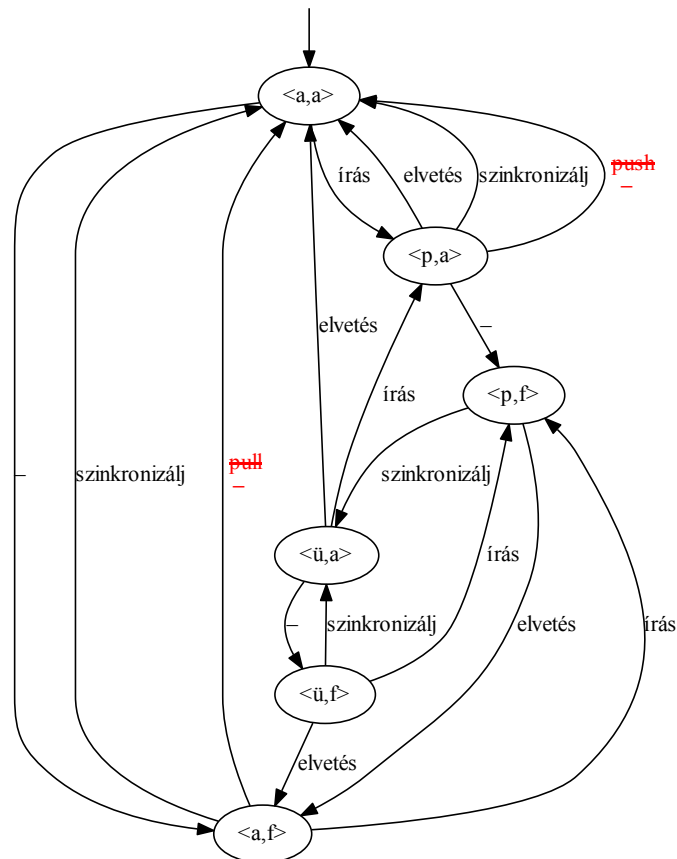


- f) Fejtsük ki a teljes összetett állapotgépet a vegyes szorzatban részt vevő két állapotgép alapján.

Megoldás

Az összetett állapotgép állapotát egy kételemű állapotvektor ($\langle \text{kliens}, \text{szerver} \rangle$) írja le. A kezdőállapotból ($\langle \text{azonos}, \text{azonos} \rangle$) indulva minden eseményre rögzítjük egy táblázatban, hogy mi a következő állapot. Randevú eseményen csak akkor lép az összetett állapotgép, ha mindkét állapotgép tud lépni az adott eseményre. (A táblázatban * jelöli a hurokéleket.)

	aszinkron (egyedi) események		közös (osztott) események	randevú események		spontán átmenet
	elvetés	írás		push	pull	
$\langle \text{azonos, azonos} \rangle$	*	$\langle p, a \rangle$	*	–	–	$\langle a, f \rangle$
$\langle \text{piszkos, azonos} \rangle$	$\langle a, a \rangle$	*	$\langle a, a \rangle$	$\langle a, a \rangle$	–	$\langle p, f \rangle$
$\langle \text{azonos, frissült} \rangle$	*	$\langle p, f \rangle$	$\langle a, a \rangle$	–	$\langle a, a \rangle$	–
$\langle \text{piszkos, frissült} \rangle$	$\langle a, f \rangle$	*	$\langle \ddot{u}, a \rangle$	–	–	–
$\langle \text{ütközés, azonos} \rangle$	$\langle a, a \rangle$	$\langle p, a \rangle$	*	–	–	$\langle \ddot{u}, f \rangle$
$\langle \text{ütközés, frissült} \rangle$	$\langle a, f \rangle$	$\langle p, f \rangle$	$\langle \ddot{u}, a \rangle$	–	–	–



Vegyük észre, hogy az összetett automatán pirossal jelölt éleken a belső randevú események helyett spontán átmenet szerepel, mivel a randevú célja a két állapotgép szinkronizálása volt, amely az összetett automatában megvalósul. (Az átláthatóság érdekében a hurokélek nem szerepelnek a gráfon.)

- g) (Kiegészítő feladat.) Ebben a modellben a szerver és a kliens közvetlenül figyelembe tudják venni egymás belső állapotát, és a szinkronizáció is pillanatszerűen végbemegy közöttük. Egy valódi elosztott rendszerben azonban üzenetváltással kell a kliens és a szerver közötti kommunikációt megvalósítani; a küldés és a válasz megérkezése között pedig huzamosabb idő eltelhet. Gondoljuk végig, hogy lehetne finomítani a modellt, hogy ezeket a részleteket is tükrözze!

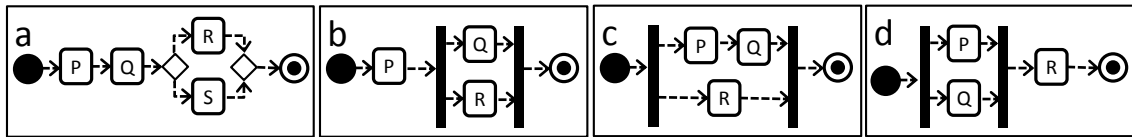
Megoldás

Otthoni feladat.

2. Folyamat lefutása

Egy folyamat végrehajtása során az összes lépést megfigyeltük. A következő eseménysor bekövetkeztét észleltük:

Folyamat indul, P elkezdődik, P befejeződik, Q elkezdődik, R elkezdődik, Q befejeződik, R befejeződik, Folyamat befejeződik.



Az a, b, c, d folyamatmodellek közül melyek lehetnek helyes modelljei a rendszernek?

Megoldás

A b és a c folyamatmodellek. Ahol nem illeszkedik, ott mutassuk meg, hogy az eseménysor hol tér el a folyamatmodellről.

3. Vezérlési folyamat (forráskód alapján)

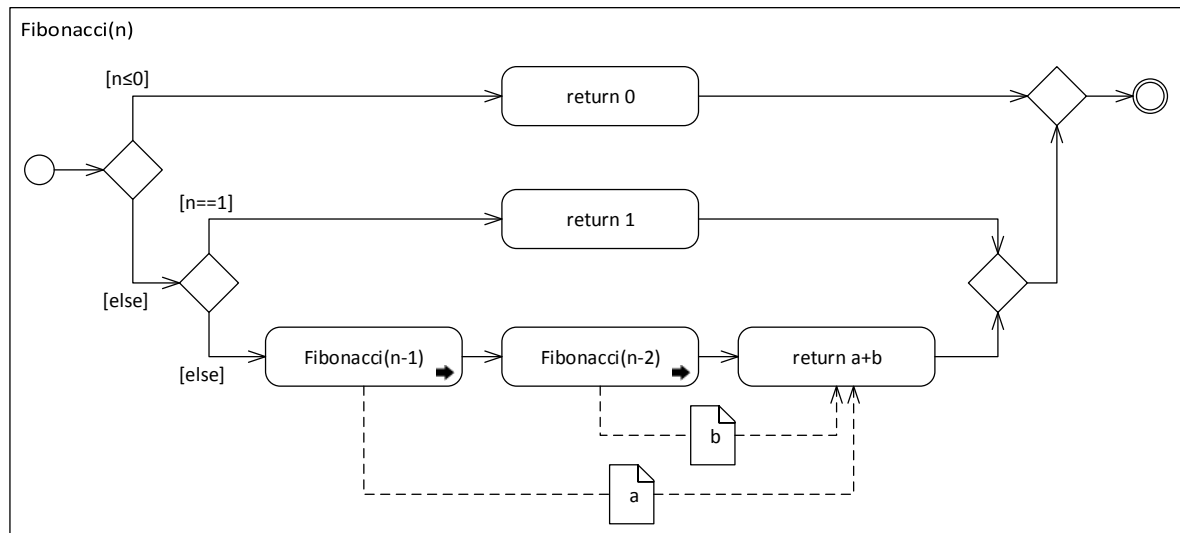
Tekintsük az alábbi C nyelvű függvényt.

```
1 unsigned long long fibonacci(int n)
2 {
3     if (n <= 0) {
4         return 0;
5     } else if (n == 1) {
6         return 1;
7     } else {
8         unsigned long long a = fibonacci(n - 1);
9         unsigned long long b = fibonacci(n - 2);
10        return a + b;
11    }
12 }
```

a) Milyen vezérlési folyamatot határoz meg a függvény?

Megoldás

A rekurzív hívást egy „hívás” elemmel ábrázoljuk (a doboz sarkában van egy nyíl).



b) Ellenőrizzük, hogy jólstrukturált-e ez a folyamat!

Megoldás

A jólstrukturáltság definícióját lásd a jegyzetben.¹ A folyamataink jólstrukturáltak. Ezt úgy tudjuk megmutatni, hogy az egyes tevékenységektől indulva, belülről kifelé haladva megmutatjuk minden részfolyamatra, hogy jólstrukturált. Utolsóként a teljes folyamatmodellt kell ellenőriznünk:

Egy teljes folyamatmodell jólstrukturált, ha egyetlen belépési pontja (Flow begin) és kilépési pontja (Flow end) egy jólstrukturált blokkot zár közre.

A folyamatmodellünk jólstrukturált, mert a fenti feltételeknek megfelel.

c) Azonosítsuk az adatfüggőségeket (adatáramlást) a tevékenységek között!

¹<https://ftsrg.mit.bme.hu/remo-jegyzet/folyamatmodellezes.pdf>

Megoldás

A két rekurzív hívásból megy adatáramlás az összeg return lépésébe (az ábrán szaggatott vonalakkal). A lényeg, hogy a második híváshoz voltaképp nem kell az első eredménye.

- d) Ha a programozási nyelv vagy a futtatókörnyezet megengedi, hol van lehetőség párhuzamosításra?

Megoldás

A c) feladat megoldása alapján ez triviális.

- e) (Kiegészítő feladat.) Mi biztosítja azt, hogy a függvény előbb-utóbb terminál?

Megoldás

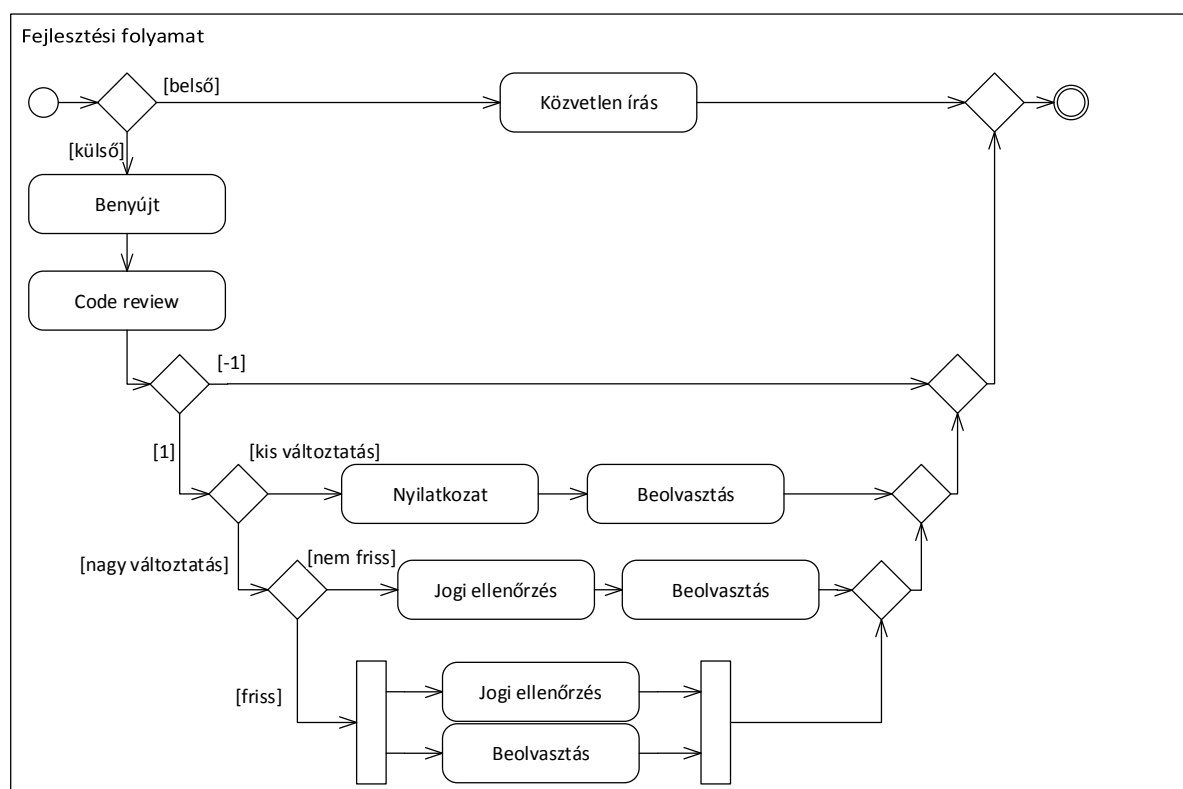
Az n változó értéke minden hívás során csökken, így előbb-utóbb teljesül az $n \leq 0$ feltétel.

4. Folyamatmodell szöveges specifikáció alapján

Egy nagy szoftveralapítvány kódtára (pl. Git) számos nyílt forráskódú szoftver fejlesztésének ad otthont. A megbízható belső fejlesztőkön kívül külsősök is gyakran küldenek be hibajavításokat vagy újonnan megvalósított képességeket. Oda kell figyelni arra, hogy a kiadott szoftverben csak jogszerűen (pl. munkaadó beleegyezésével) bekerült forráskód szerepeljen.

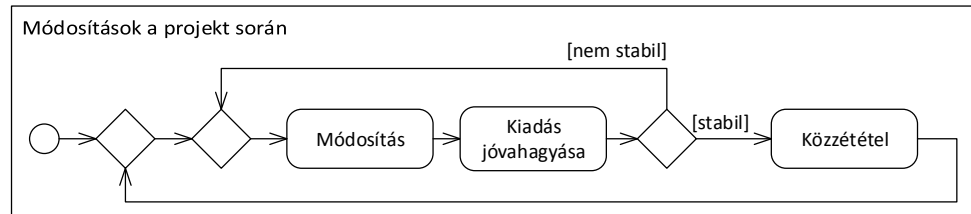
- a) Ha egy fejlesztő hozzá szeretne járulni egy projekthez az általa készített forráskóddal, akkor a saját státuszától függő lépéseket kell tennie. Belső fejlesztők közvetlenül írhatnak a kódtár adott projekt részére fenntartott területére. Külsős fejlesztőknek először átvizsgálásra (*code review*) be kell nyújtaniuk a kódjukat; ezután egy belső fejlesztőnek ellenőriznie kell azt, és utána vagy elutasítania, vagy elfogadnia. Ha a kívülről érkező kód egy bizonyos küszöbértéknél rövidebb (pl. néhány soros hibajavítás), akkor az elfogadás után a készítőjének már csak egy rövid hozzájárulási nyilatkozatot kell tennie, hogy beolvasható legyen a kódtárba.

A nagyobb lélegzetű külső hozzájárulások (pl. egy teljesen új modul beépítése) esetében azonban az elfogadást követően az alapítvány jogi osztálya egy külön adminisztratív eljárásban tisztázza a változtatások szellemi tulajdonának jogállását, és csak ennek sikeres lezárása után olvashatja be a belső fejlesztő a kódot. Frissen indított, első hivatalos kiadásuk előtt álló projekteknél itt tesznek egy kivételt: az elfogadott külső hozzájárulás kódtárba beolvasztásával nem kell megvárni ezt az adminisztratív eljárást. Készítsünk folyamatmodellt az itt leírt tevékenységekből!

Megoldás

- b) A szoftver fejlesztési projektje abból áll, hogy újabb és újabb módosításokat végeznek a forráskódon, amíg a projekt vezetése úgy nem látja, hogy a szoftver kellően stabil egy hivatalos kiadáshoz (*release*). Amikor eljött ez a pont, akkor közzétesznek egy új stabil verziót a szoftverből, majd ismét a fejlesztésen a sor, és így tovább. Készítsünk folyamatmodellt az itt leírt tevékenységekből!

Megoldás



Az *Módosítás* tevékenység alatt azt értjük, amikor a fejlesztők a kódot készítik és benyújtják az újabb kiadást.

Vegyük észre, hogy a modell csak a *folyamatra* fókuszál, nem jelennek meg benne (a szöveges specifikációban még szereplő) szereplők (aktorok).

- c) (Kiegészítő feladat.) Ellenőrizzük, hogy jólstrukturáltak-e a folyamataink!

Megoldás

A b) feladatrész megoldásának folyamatmodellje nem jólformált, mert hiányzik a *Flow end* csomópont.

- d) (Kiegészítő feladat.) Milyen viszonyban állnak egymással az a) és b) feladatokban elkészített folyamatmodellek?

Megoldás

A két folyamatmodell két külön dolognak – új forráskódok kontributálásának és új verziók kiadásának – az életútját ábrázolja egyazon rendszerben.

A kettő „nagyjából diszjunkt”, de lehetnek furcsa átlapolódások, pl. kiadási cikluson átívelő code review. Az ehhez hasonló egymásra hatások ellenőrzése nem témája ennek a gyakorlatnak (ld. *Modellek ellenőrzése* témakör).