

3. gyakorlat – Folyamatmodellek, kooperáló viselkedésmodellek – Megoldások

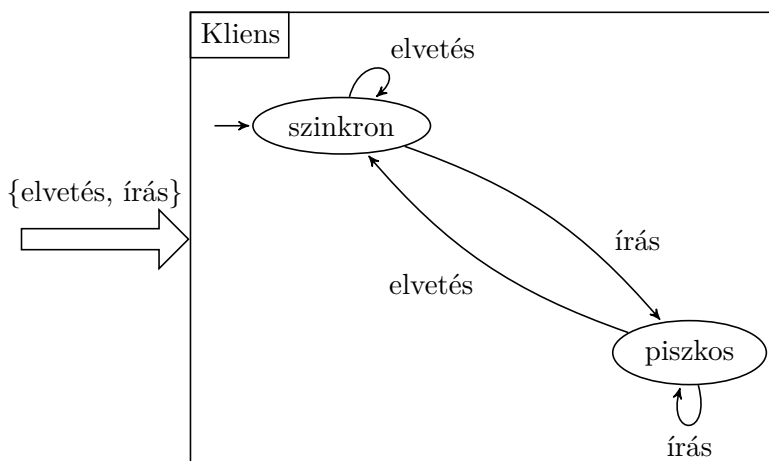
1. Felhőalapú adattárolás

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 szinkronban 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 bízta az ütközés feloldását.

- a) Modellezzük először a kliens (részleges) működését állapotgéppel! A kliens kezdetben *szinkron* 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 *szinkron* állapotba kerül.

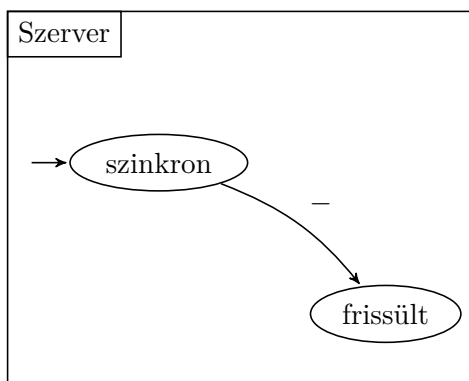
Megoldás



- b) A szerver lehetséges állapotai (csupán az adott klienssel való szinkronizációt vizsgálva) a *szinkron* 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.

Megoldás

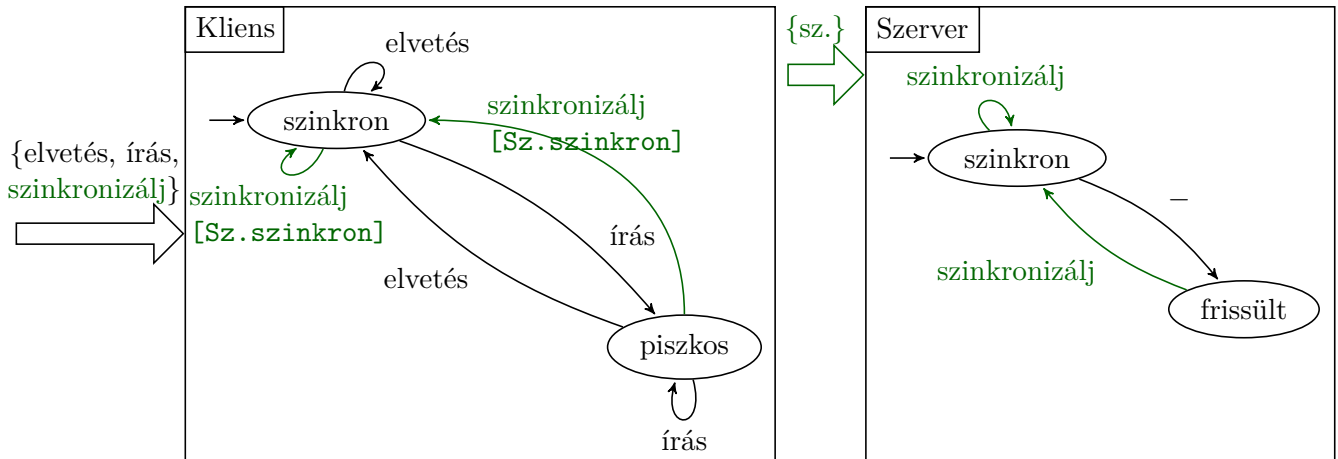
Itt annyi az érdekesség, hogy spontán állapotátmenet lesz (már ha a lokális felhasználtól érkezett bemenetekre figyelünk csak és nem modellezzünk másik klienst, pl. „kliens2”).



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

Megoldás

A Kliens automata két új átmenetére [**Szerver.szinkron**] őrfeltételt rakunk. Érdekeség, hogy ez a másik automata 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.



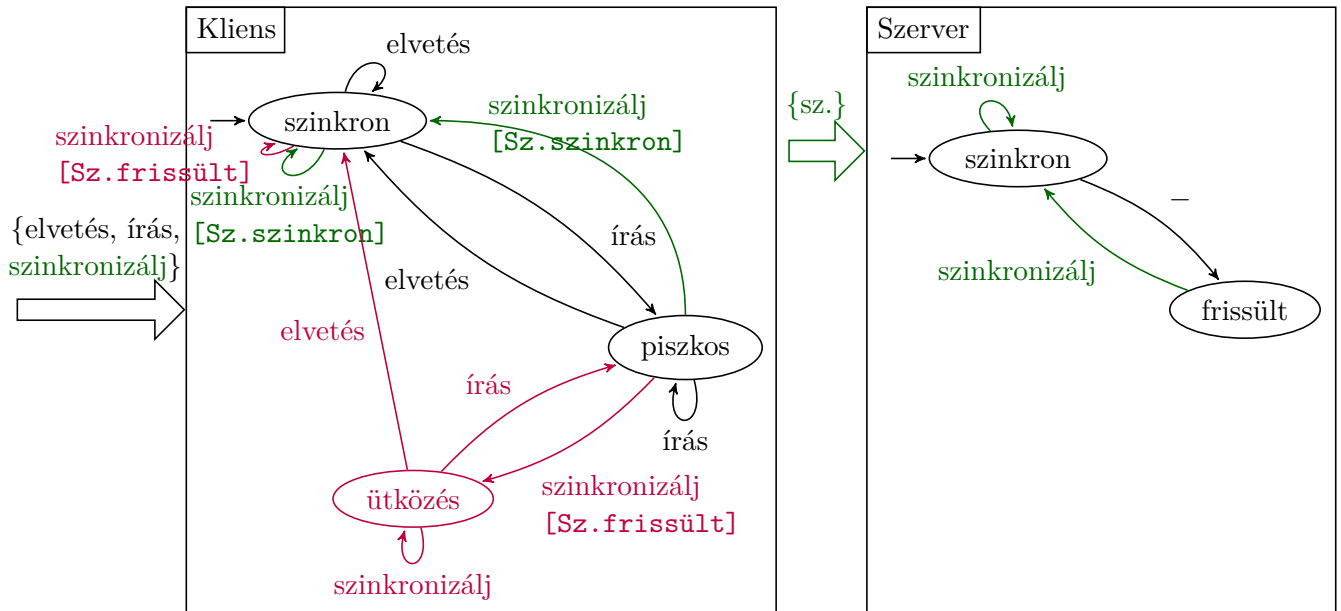
- d) Ha a szerver *frissült* állapotban van, akkor a kliensnek adott *szinkronizálj* bemenet hatására a szerver *szinkron* állapotba kerül; a kliens pedig *szinkron* állapotból nem mozdul, de *piszkos* állapotból *ütközés* állapotba megy. Mit jelent ez? Mi történjen az *ütközés* állapotban? Hol kooperál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

Megoldás

Itt egyszerre lép a két automata, tehát a *szinkronizálj* bemenet mindkét automatát befolyásolja! (Megjegyzés: 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.szinkron* állapotból két hasonló átmenet megy ki ellentétes őrfeltétellel: ([**Szerver.szinkron**], ill. [**Szerver.frissült**]), amik összevonhatóak egyetlen őrfeltétel nélküli állé (a lenti ábrán ez nem szerepel). Ez nyilván az az eset, amikor a kliens detektálja az *ütközést*. Ilyenkor a szerver *szinkron* állapotba megy, mivel önála a legutolsó szinkronizáció óta nincs újabb. A kliensen kell feloldani a konfliktust. Az *ütközés* állapotban az eseményekre például így léphetünk:

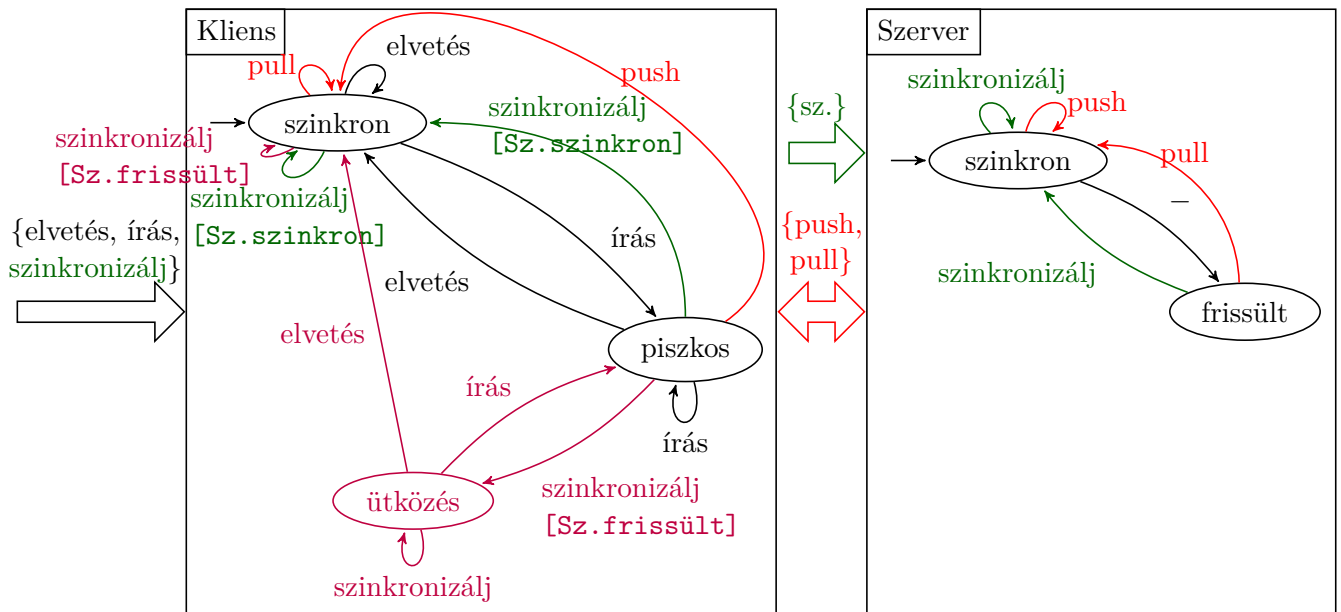
- elvetés → *szinkron*,
- írás → *piszkos*,
- szinkronizál → *ütközés*.



e) (Kiegészítő feladat.) A kliens időnként magától is szinkronizál a szerverrel, felhasználói bemenet nélkül. Mit jelent ez? Hol kooperál a két automata?

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 idézzen elő.



f) (Kiegészítő feladat.) Fejtsük ki a teljes összetett állapottérét a vegyes szorzatban résztvevő két automata alapján.

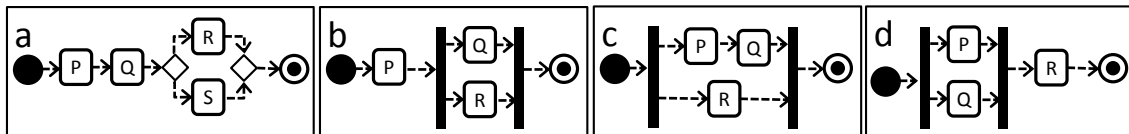
g) (Kiegészítő feladat.) Ebben a modellben a szerver és a kliens közvetlenül figyelembe tudják venni egymás belső állapotát, és a szinkronizáció is pillanatszerűen végbemegy közöttük. Egy valódi elosztott rendszerben azonban üzenetváltással kell a kliens és a szerver közötti kommunikációt megvalósítani; 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 megfigyelve a következő eseménysort észleltük: *Folyamat indul, P elkezdődik, P befejeződik, Q elkezdődik, R elkezdődik, Q befejeződik, R befejeződik, Folyamat vége.* Az alábbi 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 folyamatmodelltől.

TODO folyamatmodell-példány v. metamodell-modell van a diasor terminológiájában

3. Vezérlési folyam (forráskód alapján)

Tekintsük az alábbi C nyelvű függvényt.

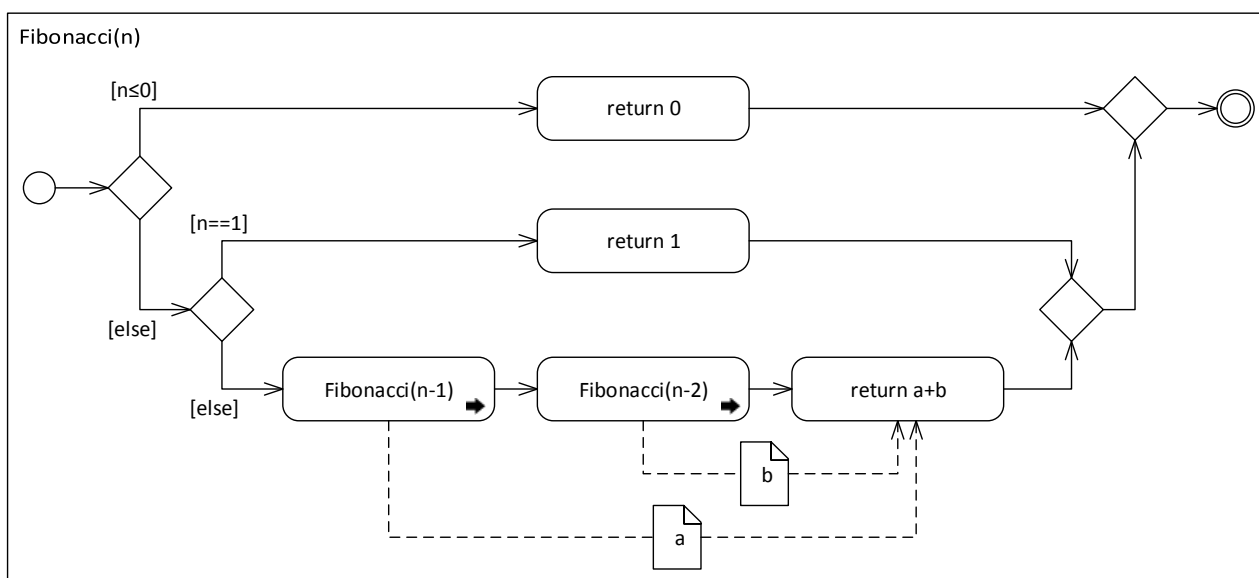
```

1 unsigned long long f(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 = f(n - 1);
9         unsigned long long b = f(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) 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.

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

Megoldás

A két rekurzív hívásból megy adatáramlás az összeg returnjébe (az ábrán szaggatott vonalakkal).

A lényeg, hogy a második híváshoz voltaképp nem kell az első eredménye.

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

Megoldás

A c) feladat megoldása alapján ez triviális.

- e) 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.

4. Folyamatmodell szöveges specifikáció alapján

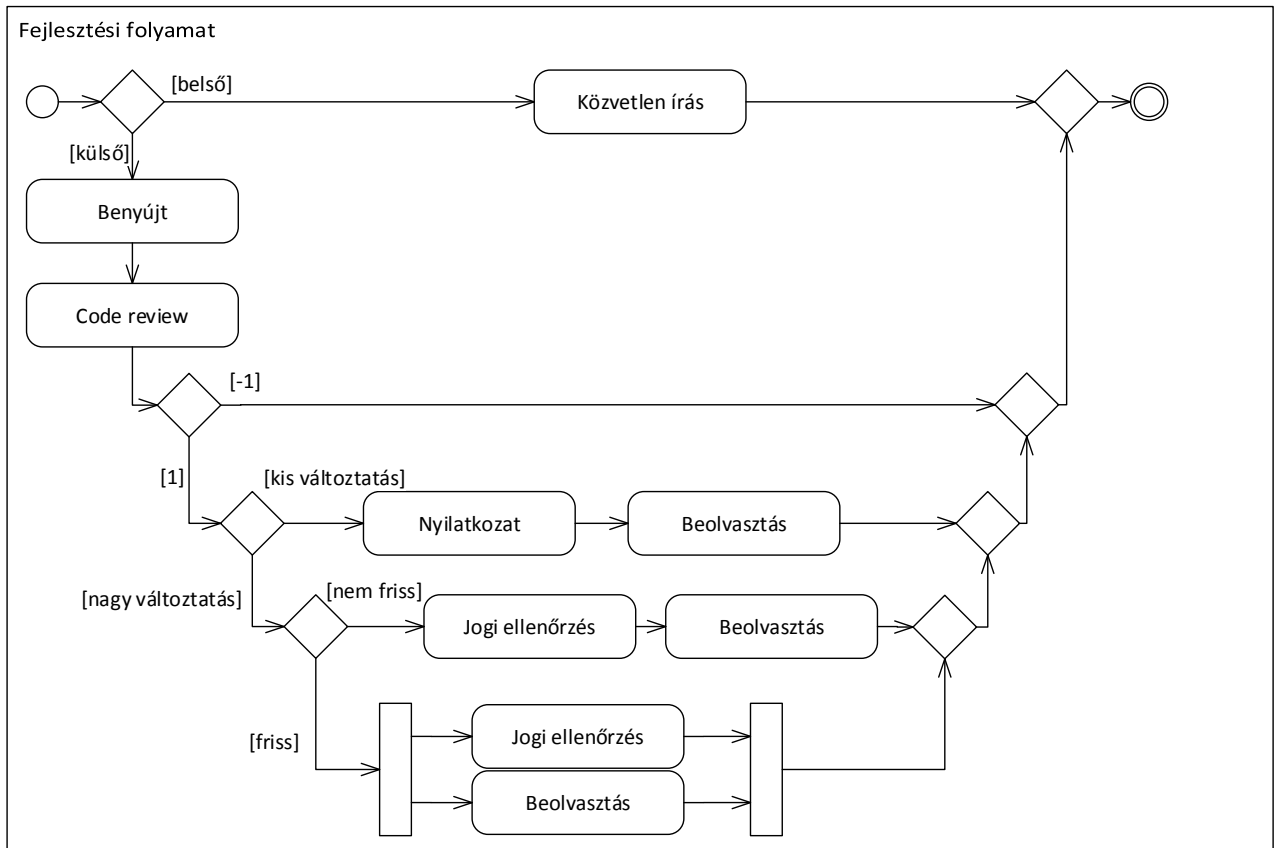
Egy nagy szoftveralapítvány kódtára (pl. Git) számos nyílt forráskódú szoftver fejlesztésének ad otthont. A megbízható belső fejlesztőkön kívül külsők is gyakran küldenek be hibajavításokat vagy újonnan megvalósított képességeket. Oda kell figyelni arra, hogy a kiadott szoftverben csak jogszerűen (pl.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

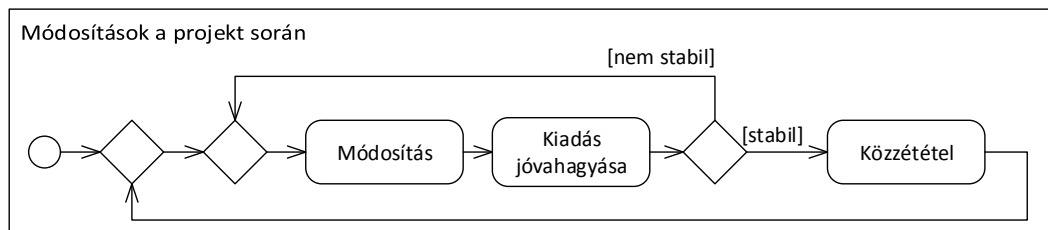
Megoldás

¹<http://docs.inf.mit.bme.hu/remo-jegyzet/folyamatmodellezes.pdf>



- 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*). Ezen a ponton 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) Milyen viszonyban állnak egymással a fenti részfeladatokban elkészített folyamatmodellek?

Megoldás

Két külön dolognak az életútja egyazon rendszerben (lehetnek furcsa átlapolódások, pl. kiadási cikluson átívelő code review).

TODO: a gyakorlatban ezeknek lehet egymásra hatása, ennek ellenőrzése nem témája ennek a gyakorlatnak (ld. *Modellek ellenőrzése* témakör).

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

Megoldás

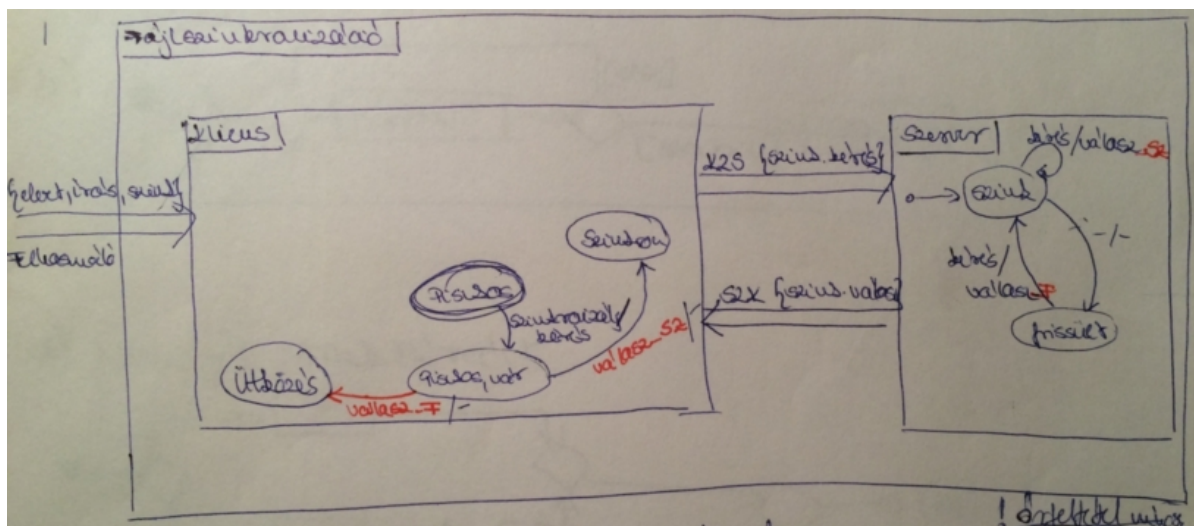
A b) feladatrész megoldásának folyamatmodellje nem jólformált, mert hiányzik a *Flow end* csomópont.

5. Felhőalapú adattárolás (részletesebben)

Az első feladatban modellezett klienst és szervert alaposabb vizsgálatnak vetjük alá, figyelembe véve, hogy hálózaton keresztül, üzenetek segítségével tartják a kapcsolatot. Ami az előző, absztrakt modellben a két automata egyidejű (atomi) közös állapotátmeneteként jelent meg, az valójában időbeli kiterjedéssel rendelkező, üzenetek küldésével és fogadásával megvalósuló kommunikációs tevékenység.

- a) Készítsünk adatfolyamhálót, amelynek a kliens és a szerver a csomópontjai! Az adatfolyamhálóban FIFO, kapacitáskorlát nélküli, pont-pont csatornákat használunk (az előadáson bevezetett módon). Határozzuk meg az adatfolyamháló bemenetén, illetve a két csomópont közötti interfészen érvényes tokeneket és a jelentésüket!

Megoldás



A felhasználói input ugyanaz. A kliens és a szerver között megy oda egy kérés, vissza egy válasz (első körben). Érdekes megkérdezni, hogy értik-e miért nem közvetlenül a felhasználó szinkronizálj üzenete jut el a szerverre, és hogy világos-e, mikor kommunikálna a két komponens.

- b) Adjuk meg a csomópontok belső viselkedésének egy lehetséges modelljét állapotgráf formalizmussal! Az állapotgráfot írjuk fel táblázatos formában is.

Megoldás

A szerver értelemszerű. Kliens: minden őrfeltétel vagy szinkron lépés esetén először egy kérést küld, majd várakozó állapotba megy. A várakozó állapotból a válasz beérkezésekor lép ki (és ha közben semmi mást nem csinál, akkor visszaabsztrahálva valóban pillanatszerű, atomi szinkron művelet lesz). Ha őrfeltétel volt, az új modellben hova lép ki a válasz beérkezésekor? Hoppá, ez nem derül ki → finomítsuk a válasz tokenet! Lesz válaszSZ (ha a szerver szinkronban van) és válaszF (ha a szerver frissítve állapotban volt), ez alapján a kliens már egyértelműen tud dönteni, de persze a szervert hozzá kell igazítani.

- c) Hogyan modellezhető az üzenetek belső struktúrája?

Megoldás

Itt egy rövid pár mondatos utalást lehet tenni, hogy az előző tokenfinomítás megfelel a *strukturális modellezés* előadásban használt altípus fogalomnak; az ott tanult eszköztár egyéb elemei is hasznosak lehetnek, pl. a kérés/válasz üzeneteknek lehet egy verziószám attribútuma, meg tartalmazhatnak fájlt.