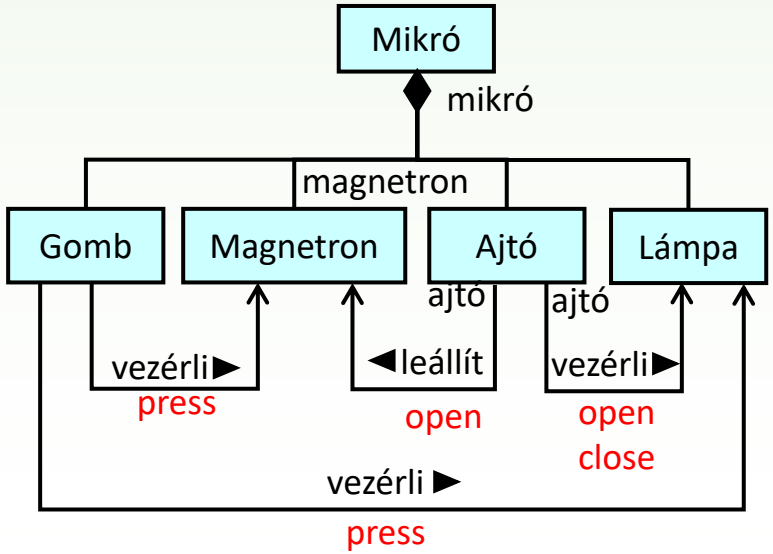
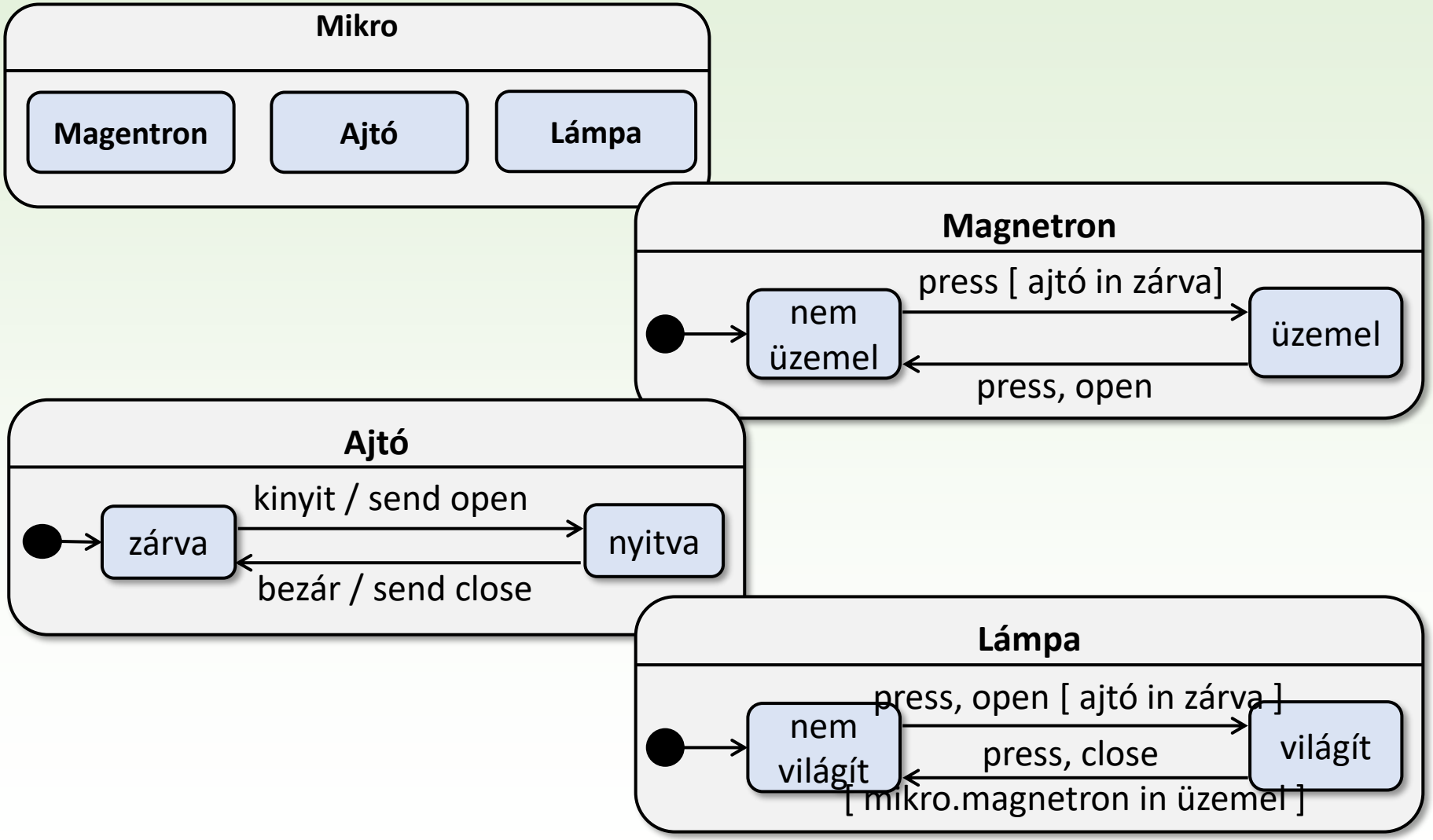
Mikrohullámú sütő

- Egy mikrohullámú sütő meghatározó elemei az ajtó, a lámpa, egy vezérlő gomb és a magnetron. A vezérlő gomb megnyomásával indítjuk el a magnetront, feltéve, hogy az ajtó csukva van, és ilyenkor a lámpa is világítani kezd. A magnetron működését vagy a vezérlő gomb megnyomásával tudjuk leállítani, ilyenkor a lámpa is kialszik, vagy az ajtó kinyitásával, de ilyenkor a lámpa égve marad. Az ajtó kinyitása mindig felkapcsolja a lámpát, bezárása pedig lekapcsolja.



Mikrohullámú sütő állapotgépe

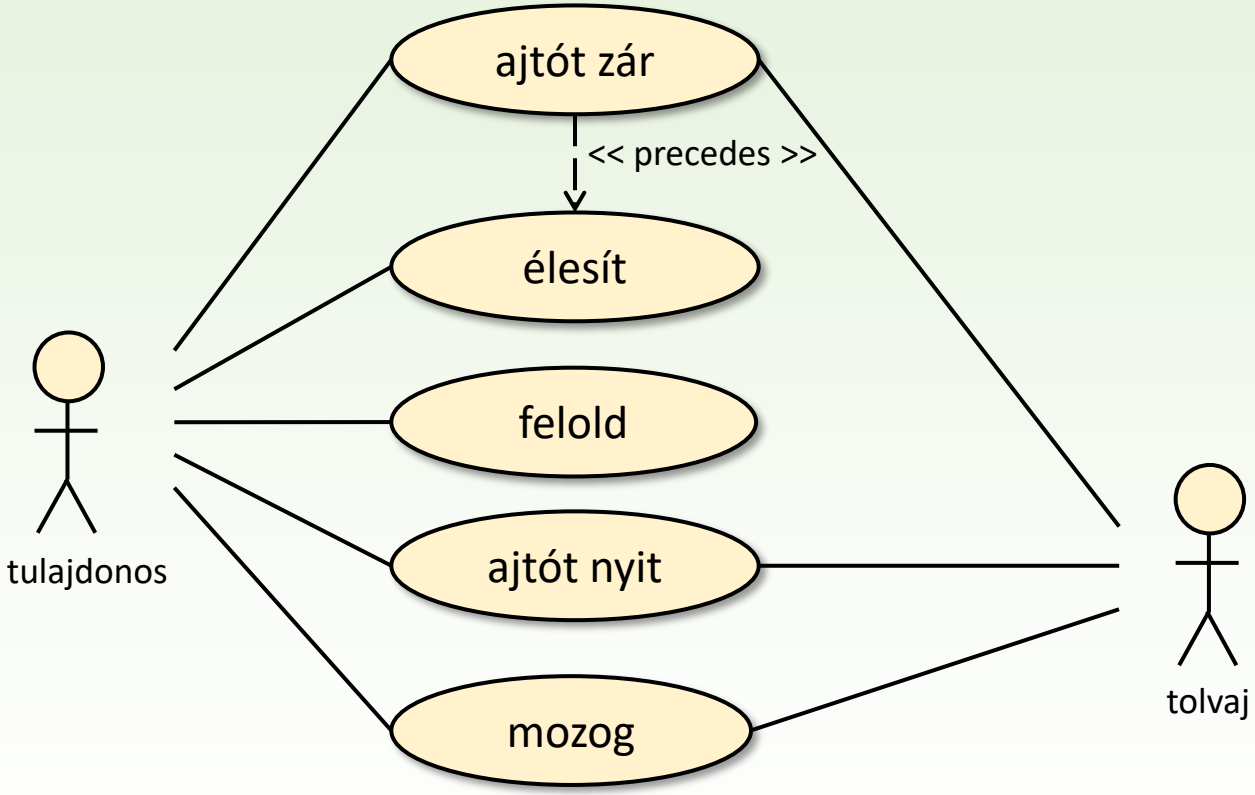


Autóriasztó

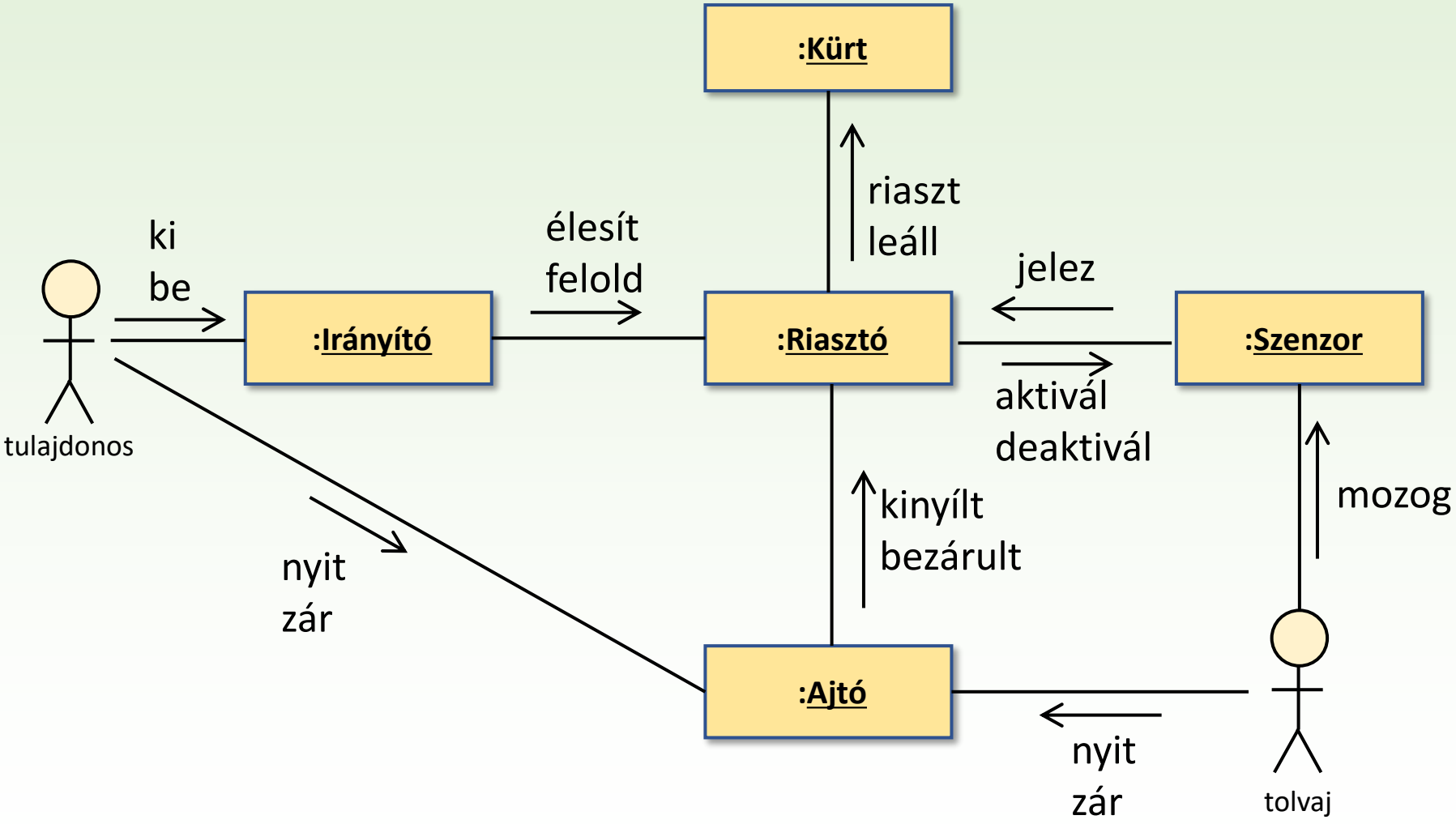
Szimuláljuk egy autóriasztó működését!

- Egy autóriasztó rendszer egy riasztó egységből, és egy mozgás-érzékelőből, és 3-5 ajtóból áll. Az ajtók nyithatók és zárhatók, a mozgásérzékelő az autóban zajló mozgások esetén jelet küld a riasztónak. A riasztót egy irányítóval lehet be-, illetve kikapcsolni. Ha a riasztó be van kapcsolva, és valamelyik ajtót kinyitják vagy az érzékelő mozgást jelez, akkor a riasztó riaszt. Ezt kikapcsolással meg lehet szüntetni. A riasztót csak akkor lehet bekapcsolni, ha minden ajtó zárva van.

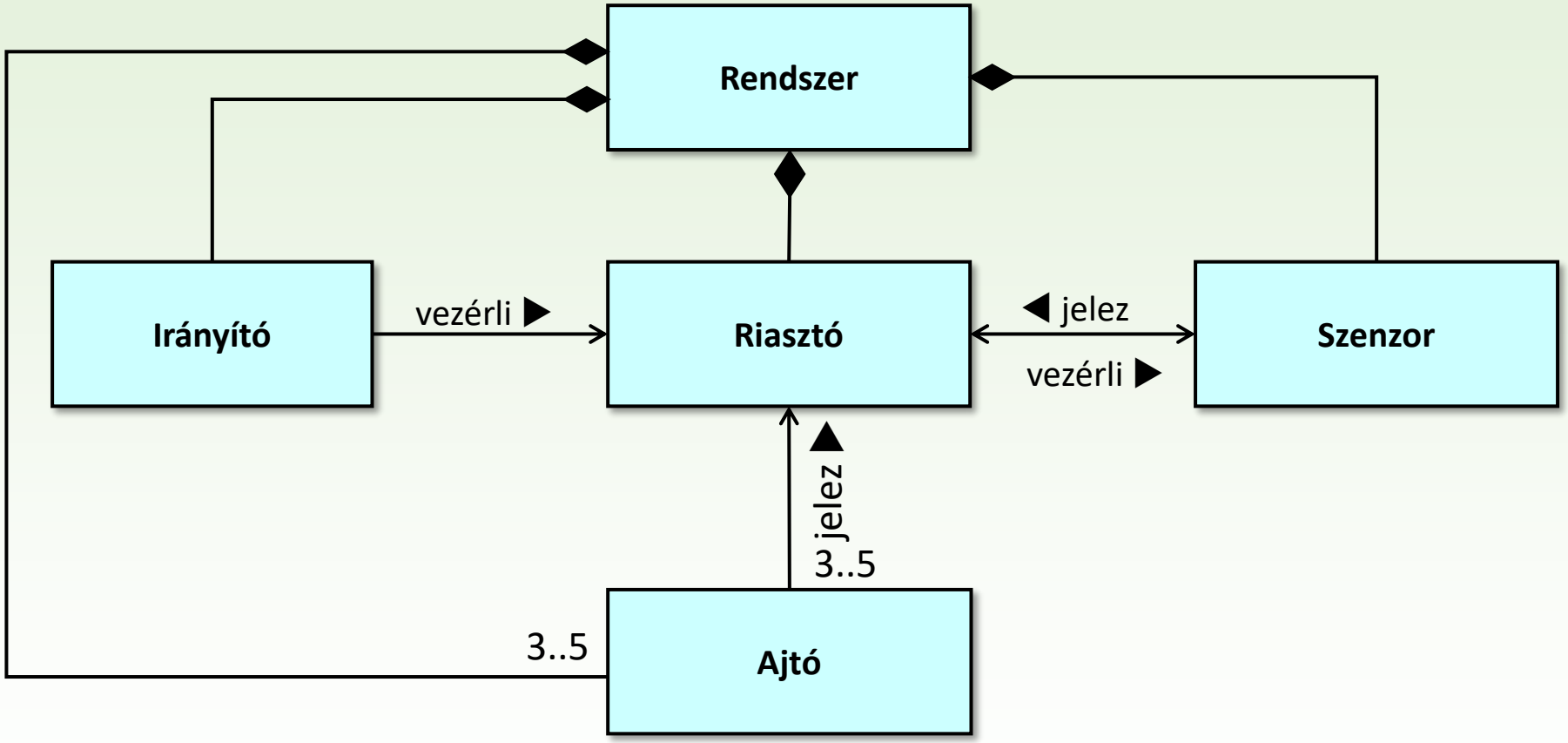
Használati eset diagram



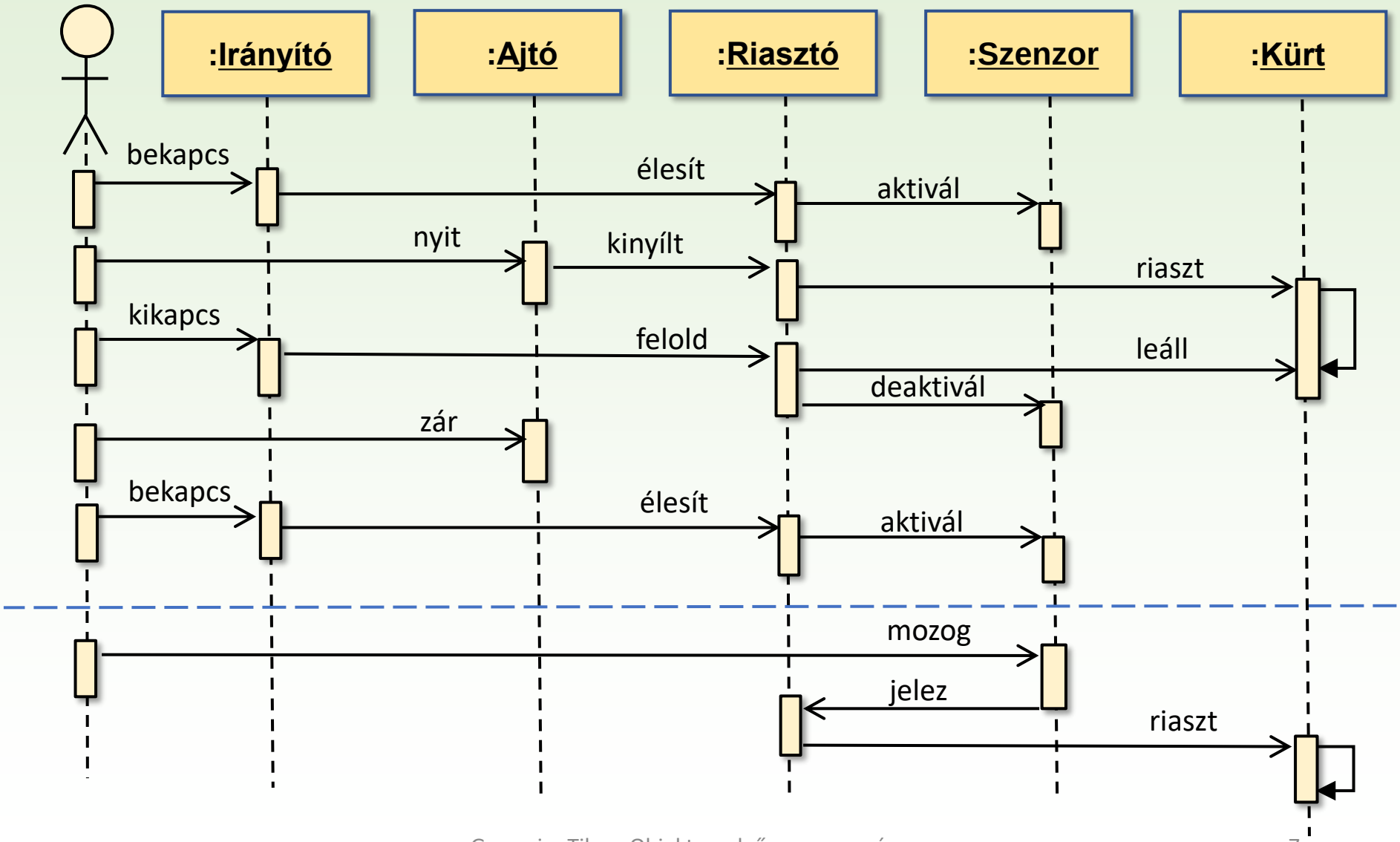
Együttműködési diagram



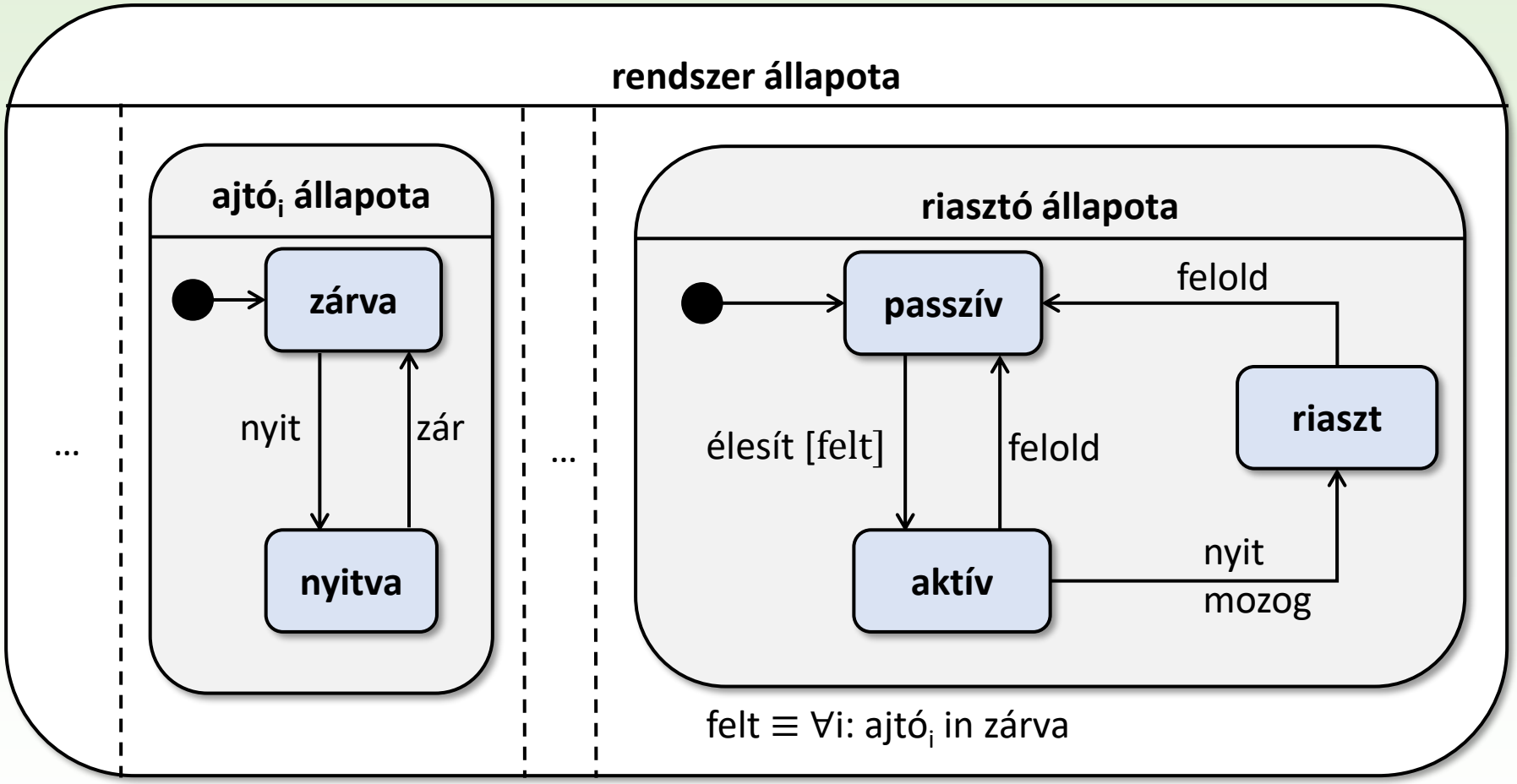
Osztálydiagram



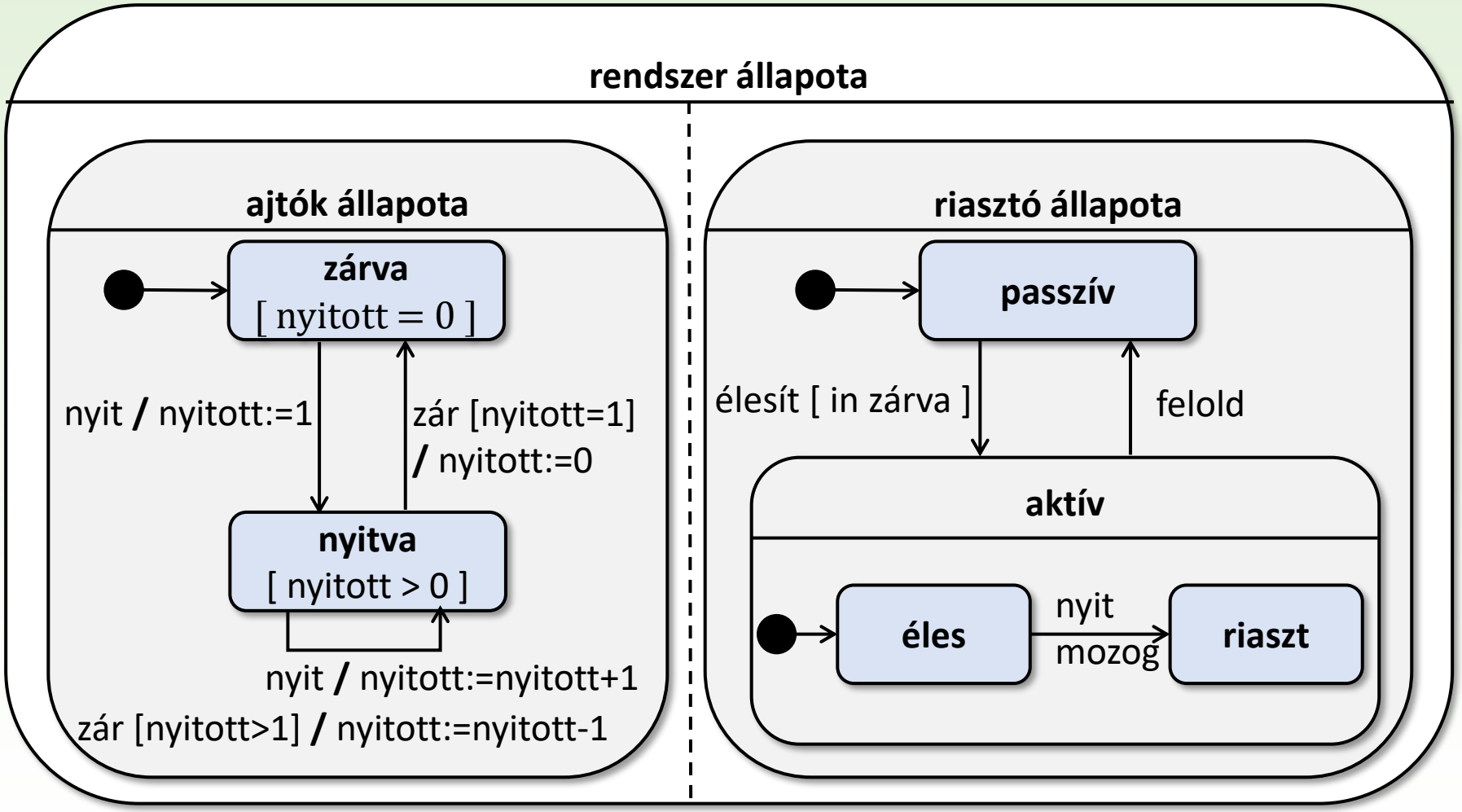
Szekvenciadiagram (2 eset)



Rendszer állapotgépe I. változat



Rendszer állapotgépe II. változat

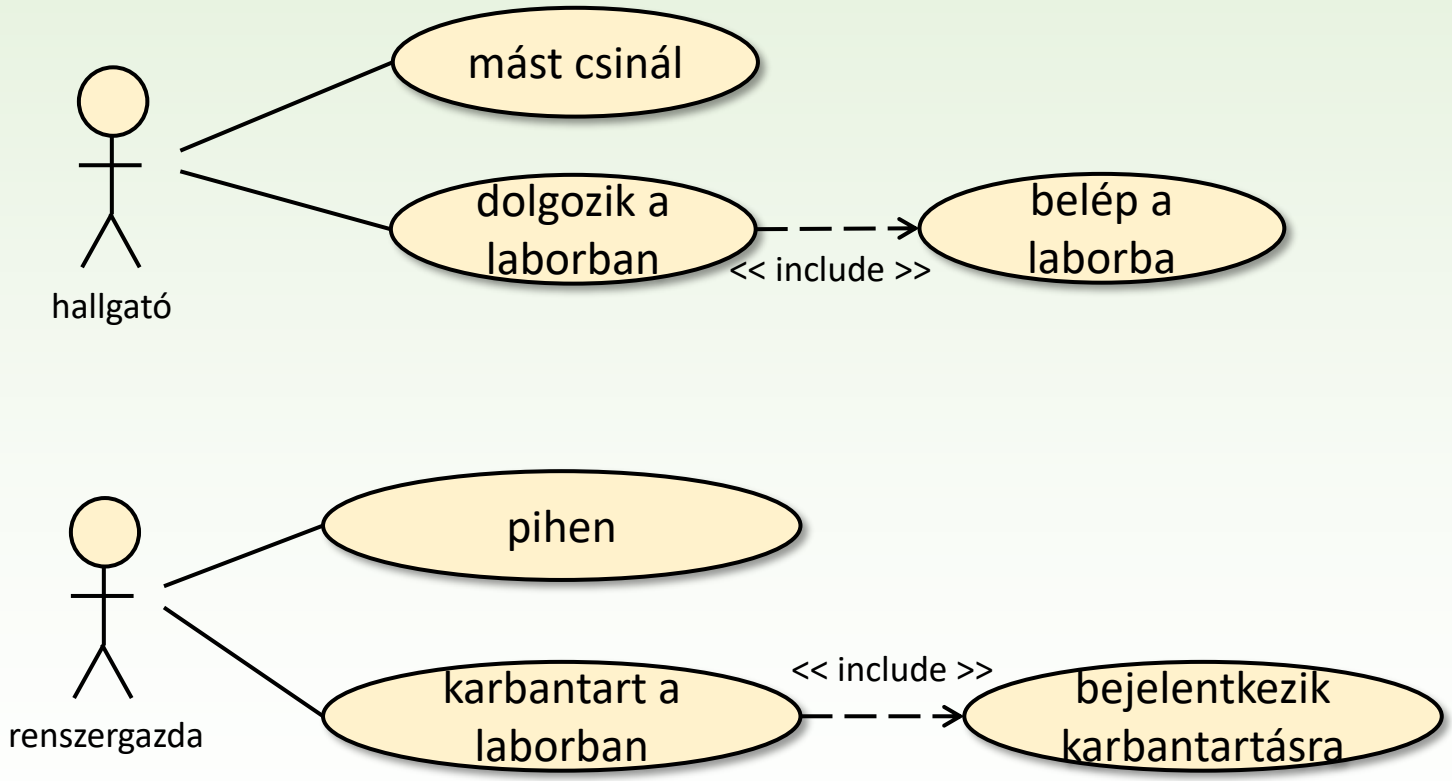


Számítógépes labor

Szimuláljuk egy számítógépes labor használatát!

- Egy számítógépes laborban számítógépek vannak.
- A labor karbantartását rendszergazdák látják el. Bármelyik rendszergazda végezhet karbantartást, de egy adott pillanatban csak egyikük. A karbantartás addig nem kezdhető meg, amíg hallgató, vagy egy másik rendszergazda tartózkodik a laborban. A rendszergazdák a karbantartás befejezése után pihennek, amíg újra nem kell karbantartást végezniük.
- Hallgatók a labort számítógép-használat céljából keresik fel. A labor előtt várakoznak, ha nincs szabad gép, vagy egy rendszergazda végez éppen karbantartást, vagy egy rendszergazda karbantartás szeretne elkezdeni. A számítógép-használat befejeztével a hallgatók elhagyják a labort, hogy mással foglalkozzanak, amíg nincs szükségük újra számítógépre.

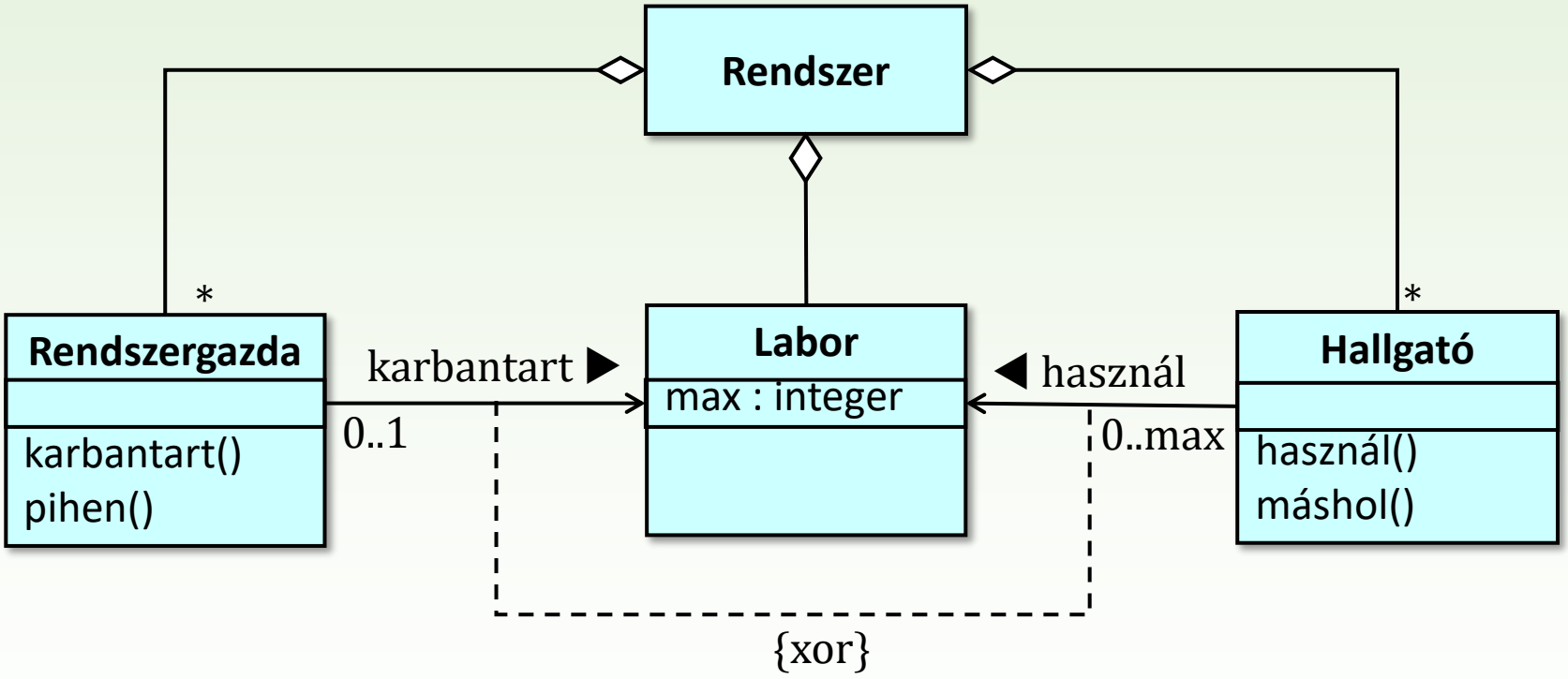
Használati eset diagram



Elemzés

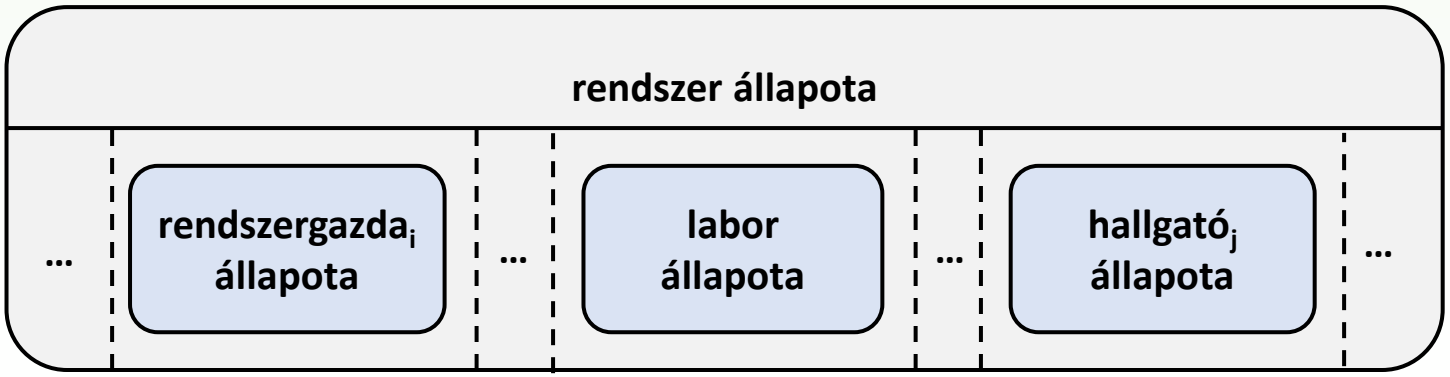
- Objektumok:
  - Teljes rendszer, amely tartalmazza az összes többi objektumot.
  - Rendszergazdák, akik felváltva a laborban és azon kívül tevékenykednek, a laborba történő belépésüknek feltétele van.
  - Hallgatók, akik felváltva a laborban és azon kívül tevékenykednek, a laborba történő belépésüknek feltétele van.
  - Számítógépes labor, amelynek fontos tulajdonsága a benne levő számítógépek száma (*cap*), a benn tartózkodó hallgatók és rendszergazdák száma.
- Objektumok közötti kapcsolatok:
  - A rendszer része a labor, *n* hallgató és *m* rendszergazda.
  - Egyszerre legfeljebb *cap* számú hallgató használhatja a labor gépeit.
  - Egyszerre legfeljebb egy rendszergazda végezhet karbantartás a laborban.
  - A használat és a karbantartás egymást kizáró tevékenységek.

Osztályok elemzési modellje



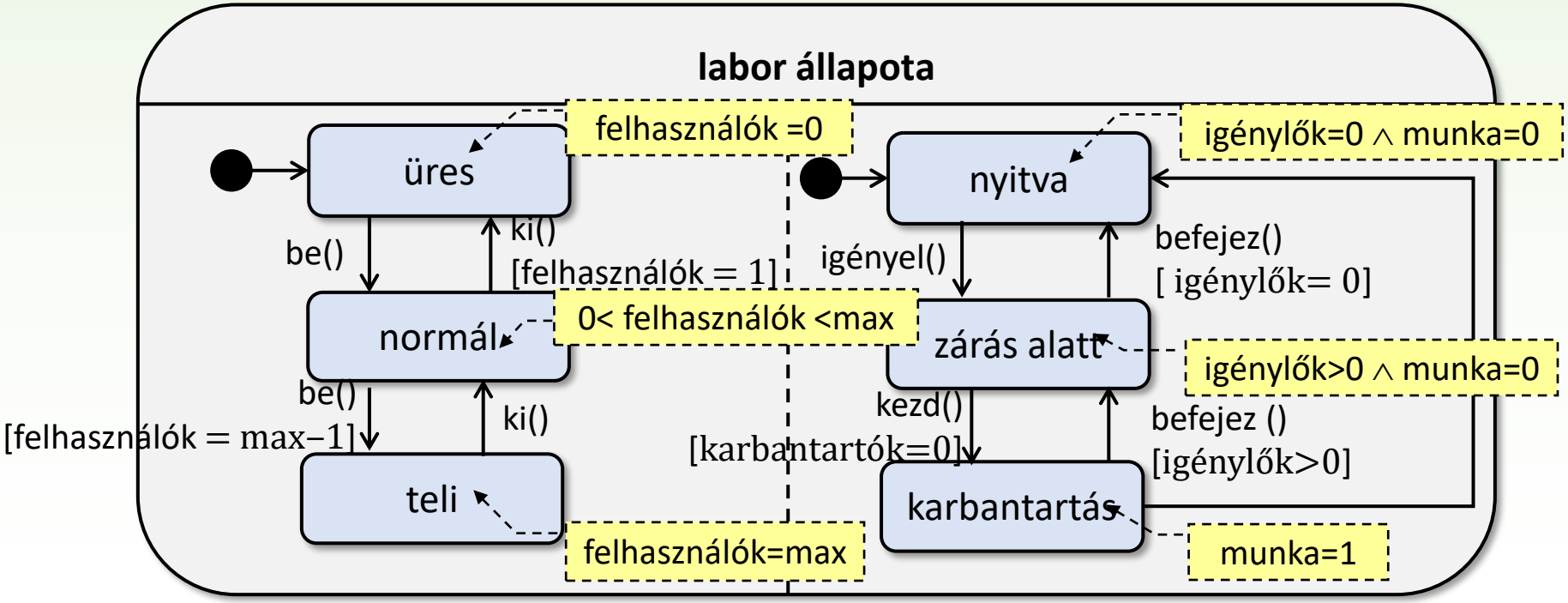
Rendszer állapotgépe

- A rendszer állapotát a rendszergazdák állapota, a hallgatók állapota és a labor állapota együttesen határozza meg.
- A rendszergazdák és a hallgatók aktív objektumok: párhuzamosan, végeznek tevékenységet, egymáshoz képest aszinkron módon változik az állapotuk, így állapotgépeiket külön szálakon kell futtatni.
- A labor passzív objektum: állapotgépe más objektumok állapotgépével szinkron módon (metódusainak hívása által) működik. Nem igényel külön szálat.



Labor állapotgépe

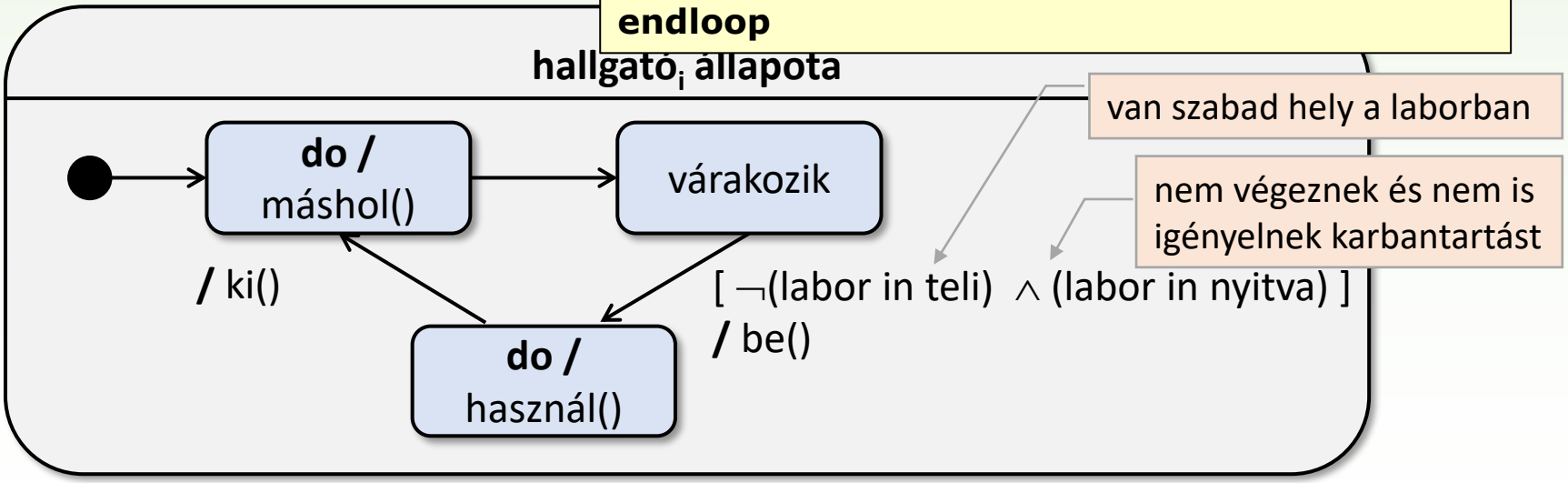
- A labor állapotgépe két ortogonális régióból áll. Működése a hallgatók szempontjából, illetve a rendszergazdák szempontjából szemlélhető.
- A labor nem küld szignálokat, az állapot átmeneteit okozó metódusokat hallgató illetve rendszergazda objektumok hívják: a labor egy passzív objektum.



Hallgatók állapotgépe

- Egy hallgató háromféle állapotban lehet, amelyek ciklikusan változnak.

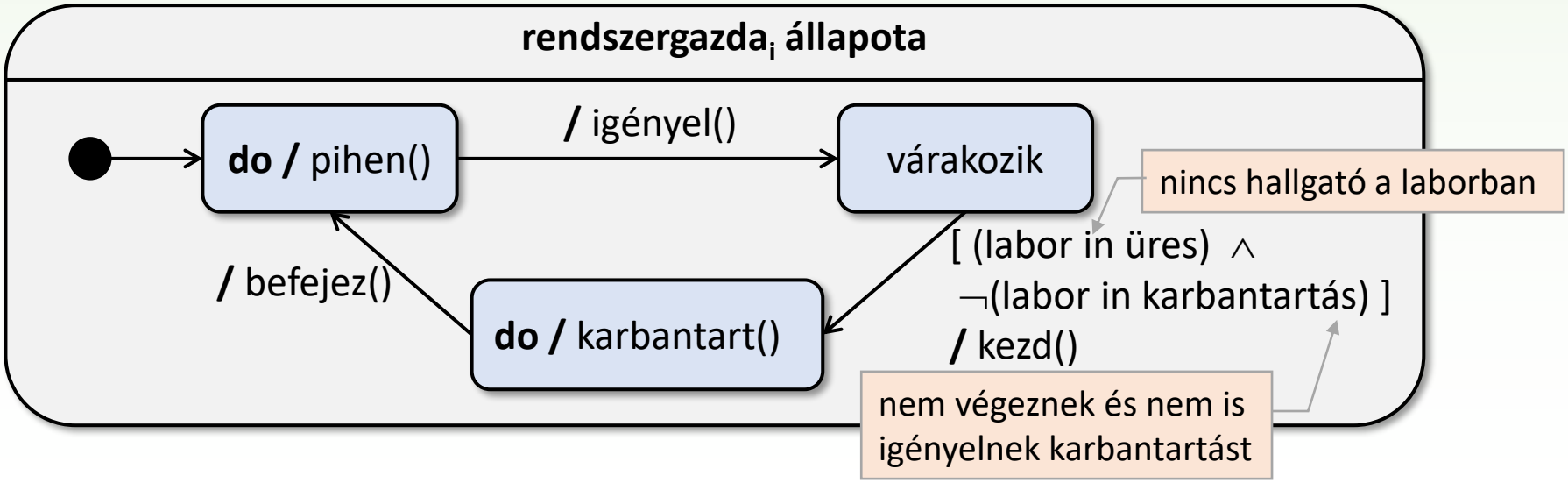
```
hallgató_j():
loop
      máshol()
      await ¬(labor in tele) ∧ (labor in nyitva) then
            be()
       end
      használ()
      ki()
endloop
```



Rendszergazdák állapotgépe

- Egy rendszergazda három állapota ciklikusan változik.

```
rendszergazda_i():
loop
      pihen()
      igényel()
      await (in labor.üres)  ∧  (¬ in labor.karbantartás) then
         kezd()
      end
      karbantart()
      befejez()
endloop
```



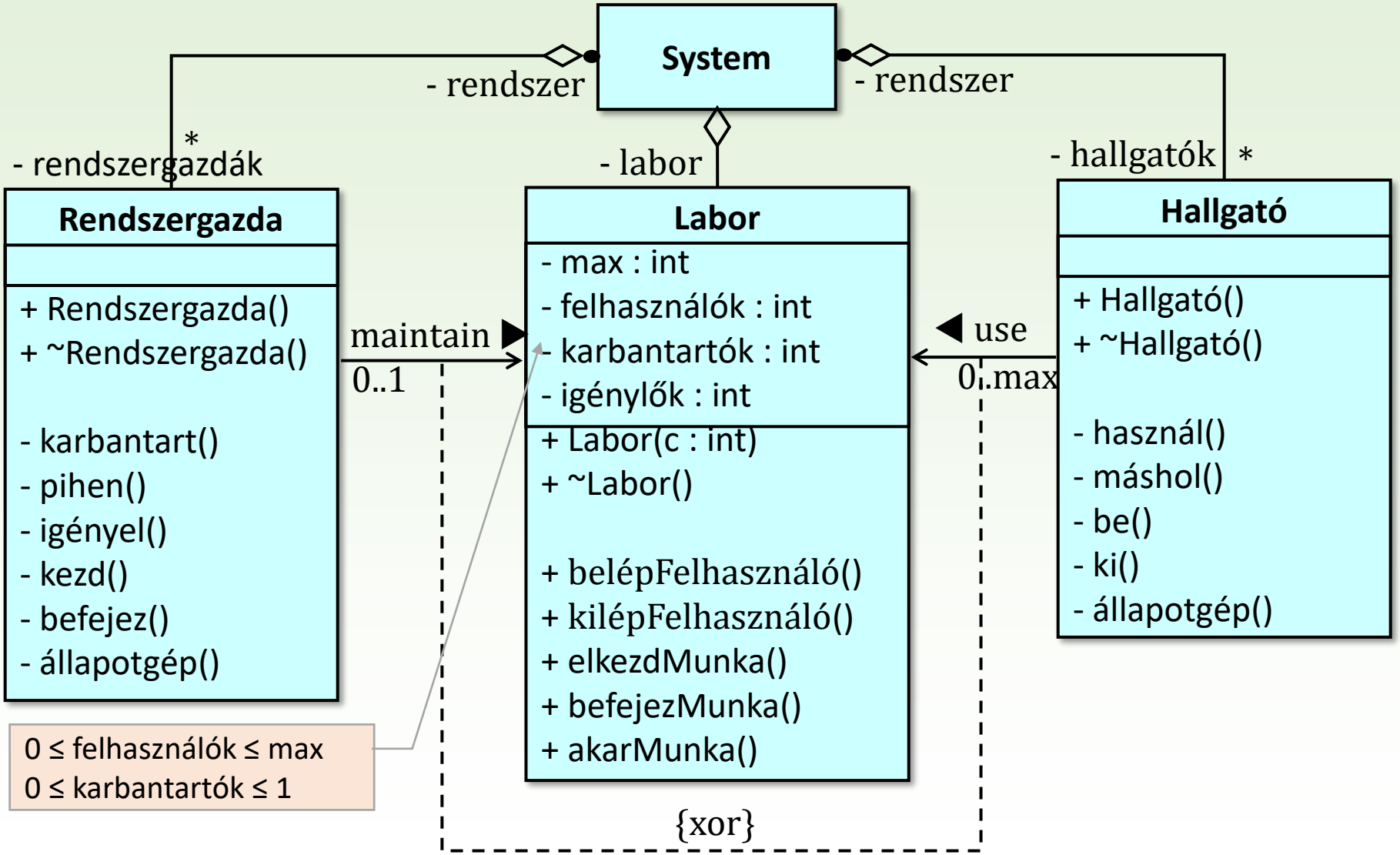
Őrfeltételek és tevékenységei

- A labor osztály definiálja a hallgatók és rendszergazdák állapotgépeinél használt őrfeltételekhez használt számlálókat (adattagok):
  - felhasználók ~ laborban levő hallgatók száma
  - karbantartók ~ laborban karbantartó rendszergazdák száma
  - igénylők ~ karbanrartásra várakozó rendszergázdák száma
- és az azokat módosító tevékenységeket (metódusok):
  - kezd(), végez(),
    igényel(), belép(), kilép()

őrfeltétel	reprezentáció
in üres ∧ ¬ in karbantartás	felhasználók=0 ∧ karbantartók=0 ∧ igénylő=0
¬ in teli ∧ in nyitva	felhasználók<max ∧ karbantartók=0 ∧ igénylő=0

tevékenységek	hatás
be()	++felhasználók
ki()	--felhasználók
igényel()	++igénylő
kezd()	--igénylő ++karbantartók
befejez()	--karbantartók

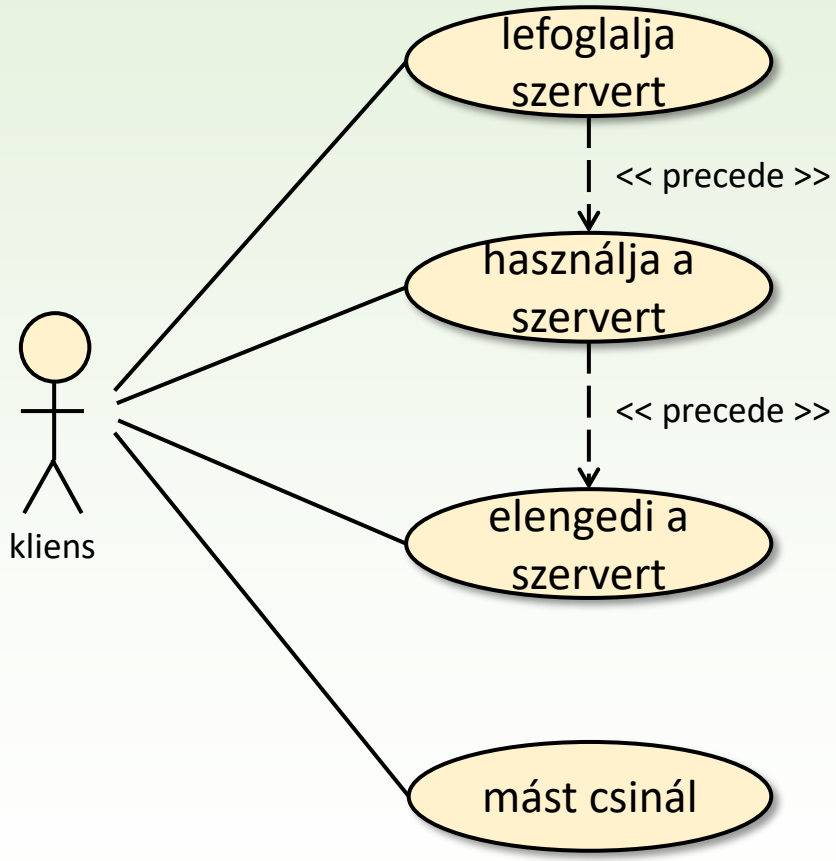
Osztályok megvalósítási modellje



Szerver-kliens

- Modellezzük azt a folyamatot, amikor különféle, egymással párhozamosan zajló tevékenységek (kliensek) ugyanazon erőforrás (szerver) szolgáltatását veszik igénybe kölcsönösen kizárásos módon.
- A kliensek működésében kétféle szakasz váltakozik.
  - kritikus szakasz: amikor a kliens a szervert kizárólagos módon használja, ekkor más kliens a szervert nem használhatja
  - nem kritikus szakasz: amikor a kliens a szervertől független tevékenységet végez, amely párhuzamosan folyik más kliensek tevékenységével.
- Mielőtt egy kliens tevékenysége kritikus szakaszba lépne, meg kell győződnie arról, hogy szabad-e a szerver. Ha nem, azaz másik kliens használja, akkor várakoznia kell a szerver felszabadulásáig.

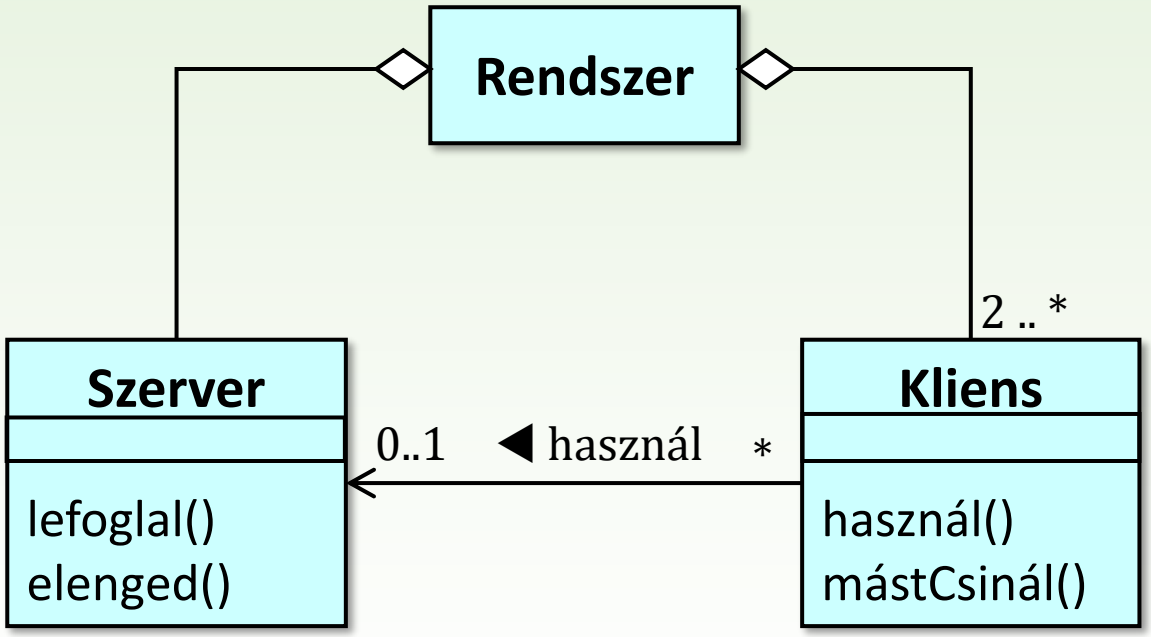
Használati eset diagram



Elemzés

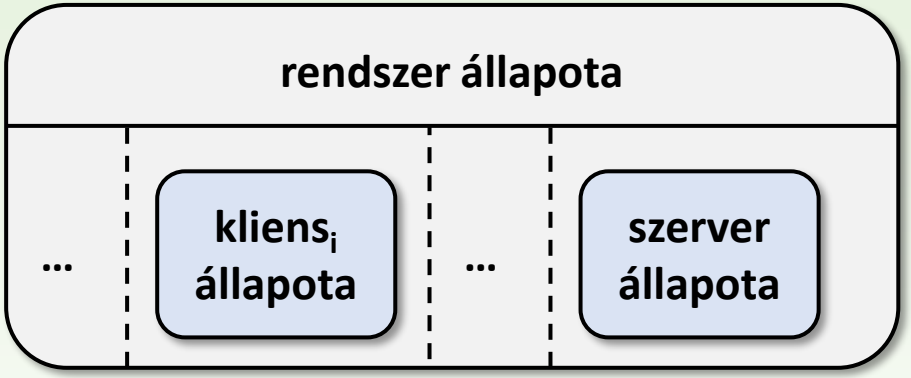
- Objektumok:
  - teljes rendszer
  - szerver
  - kliensek
- Objektumok közötti kapcsolatok:
  - A rendszer része a szerver és kettőnél több kliens.
  - Egyszerre egy kliens használhatja a szervert.
- Objektumok tevékenységei:
  - A szervert le lehet foglalni egy kliens számára, majd elengedni.
  - Egy kliens használja a szervert, vagy valami mást csinál.

Osztály diagram



Rendszer dinamikus modellje

- A rendszer állapota a szerver és a kliensek állapotaitól függ.



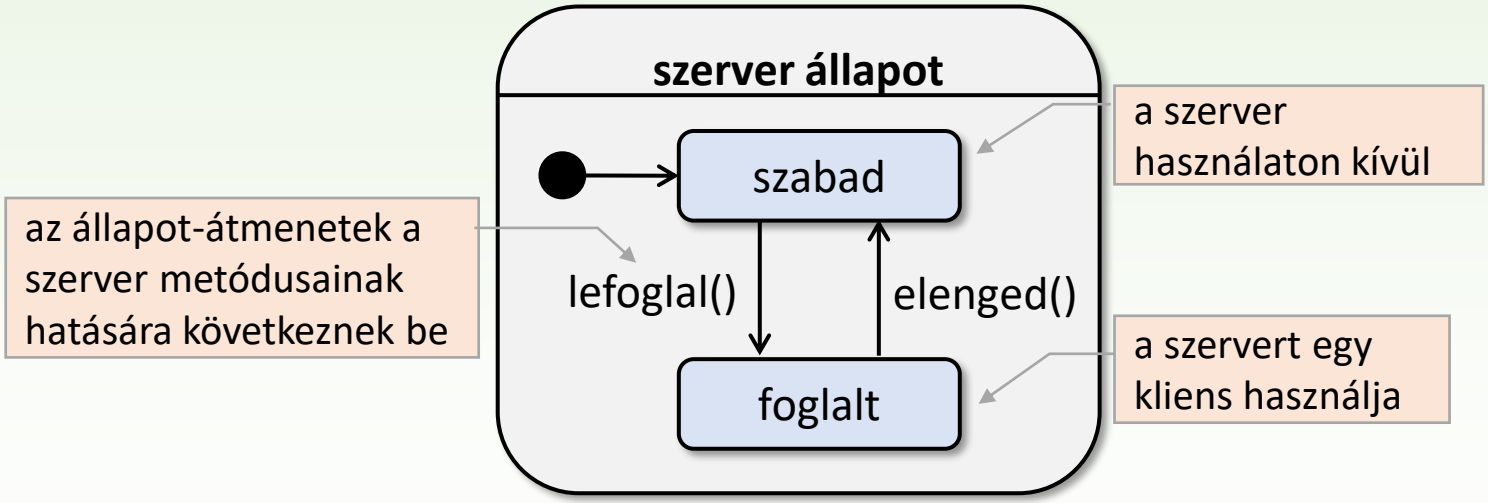
- A kliensek (aktív objektumok) egymással párhuzamosan végzik tevékenységüket, állapotaik aszinkron módon változnak.
- A szerver állapotgépe (passzív objektum) a kliensek állapotgépeivel szinkron módon működik.

```
parbegin
    kliens1()‖ ... ‖ kliensn()
parend
```

Párhuzamos programszerkezet, amely külön szálon futtatja a kliensek állapotgépeit.

Szerver állapotgépe

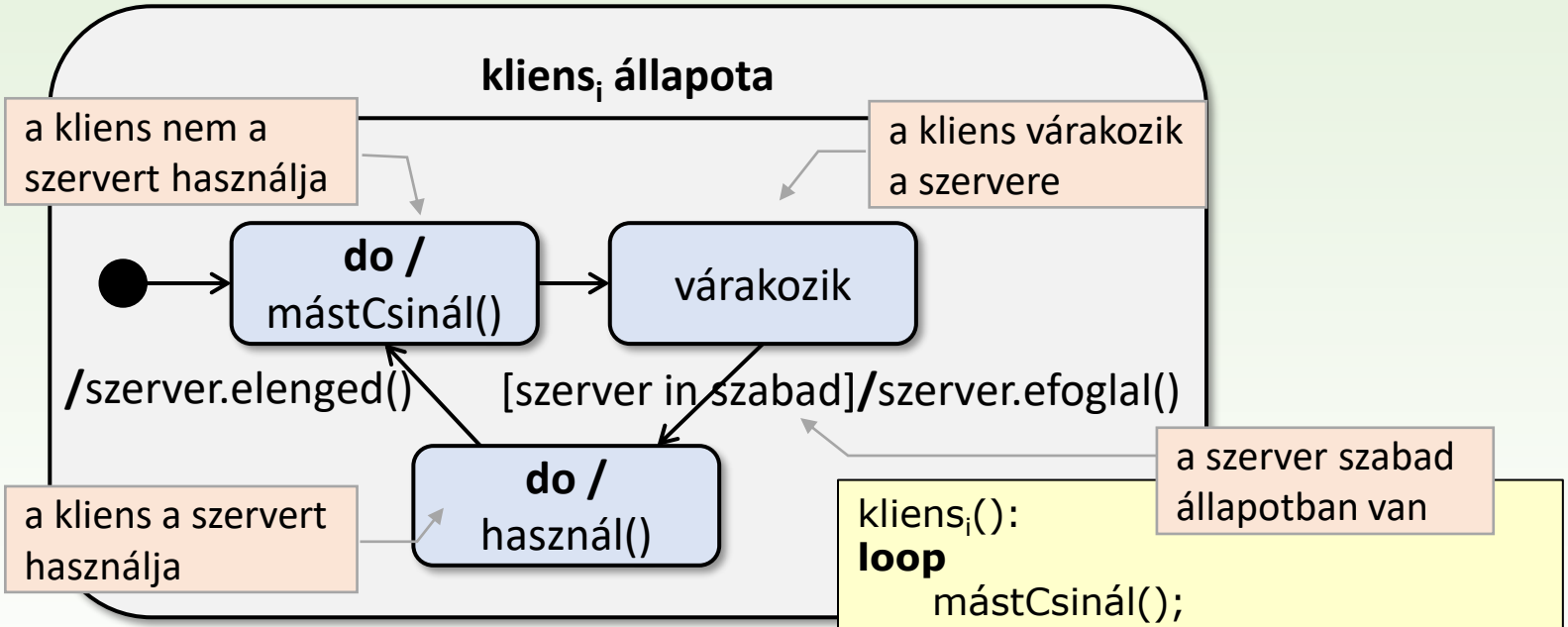
- A szerver kétféle állapotban lehet, amelyek ciklikusan változnak: egy kliens használja: foglalt vagy szabad.



a szerver egy passzív objektum: állapot-átmeneteit más objektumok szinkron metódushívásokkal vezérlik

Kliens állapotgépe

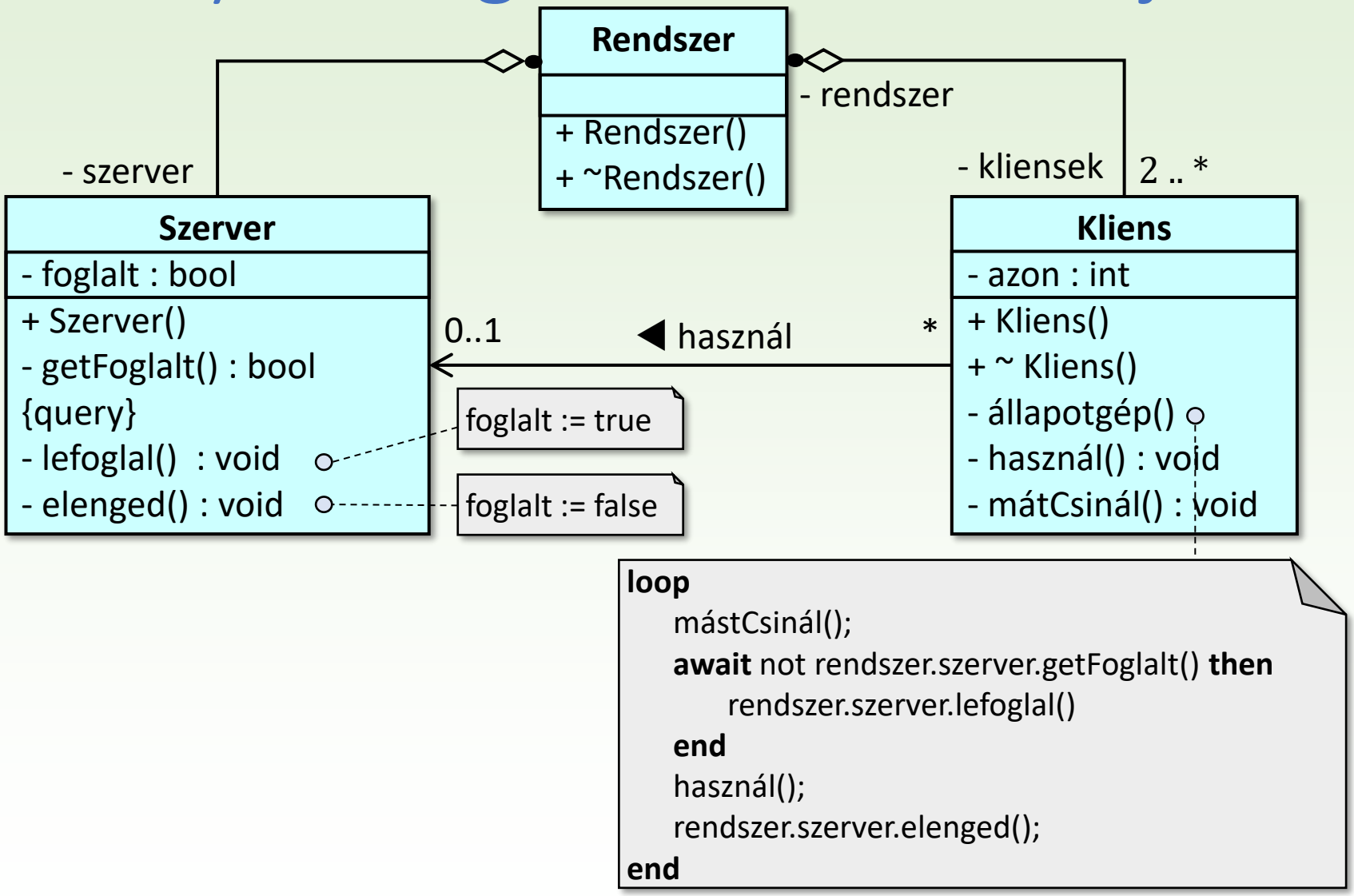
- Egy kliens háromféle állapotban lehet, amelyek ciklikusan változnak.



a kliens egy aktív objektum:
saját tevékenységének hatása okozza állapot-átmeneteit

```
kliens_i():
loop
      mástCsinál();
      await szerver in szabad then
            szerver.lefoglal()
      end
      használ();
      szerver.elenged();
endloop
```

Osztályok megvalósítási modellje

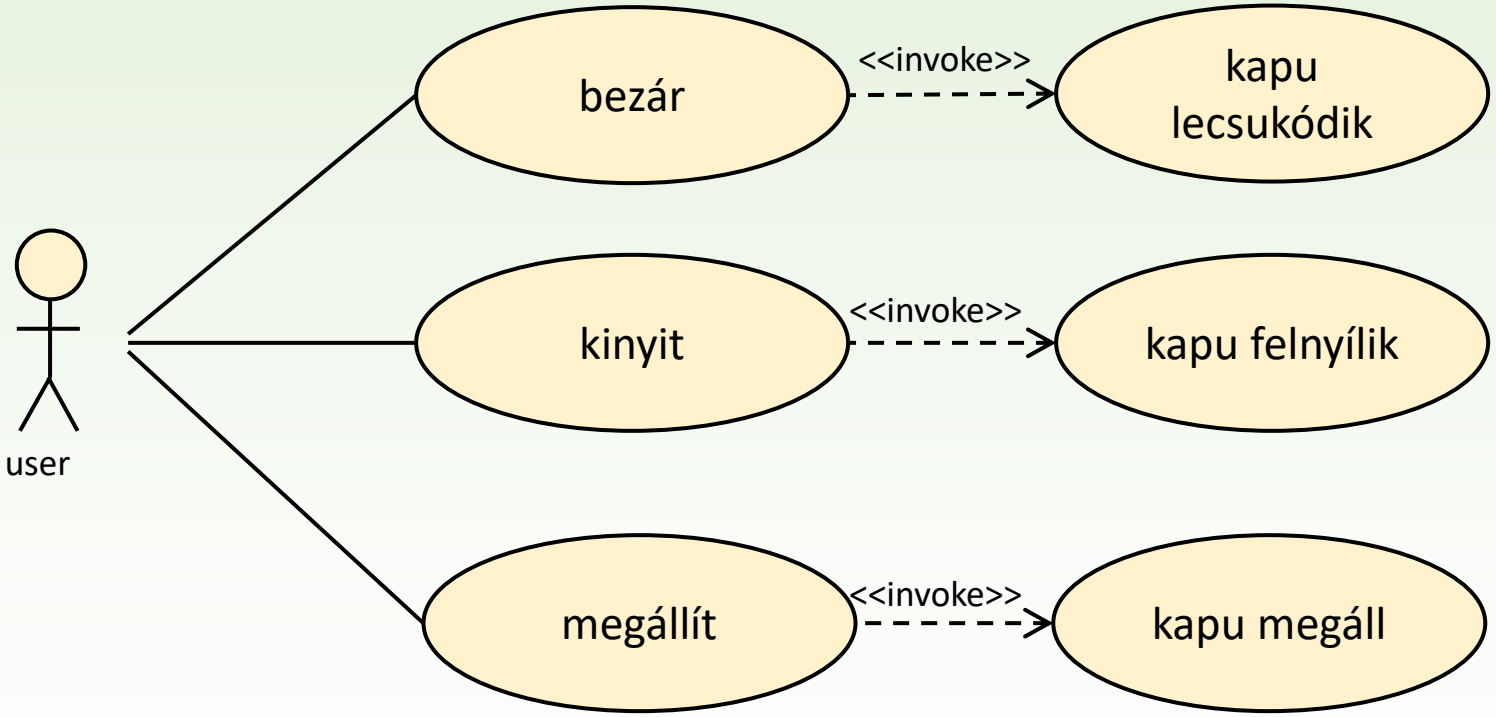


Garázskapu

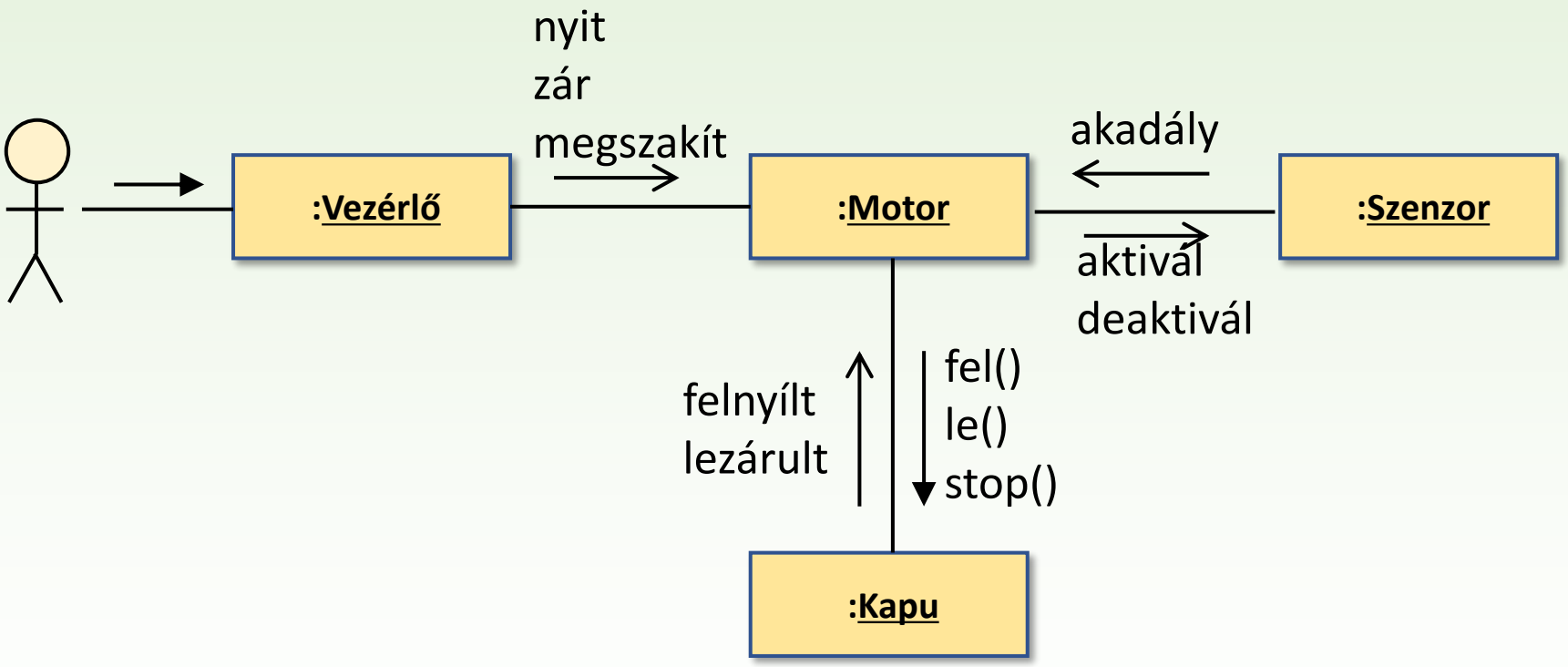
Modellezzük egy függőlegesen mozgó garázskapu működését!

- A garázskaput lezáró és felnyitó motort egy távirányító vezérli. A távirányítóval három féle parancs adható: le, fel, stop.
- A garázskapu a legutoljára adott parancs szerint működik, de magától is megáll, ha akadályba ütközik.
- Az akadályt egy olyan érzékelő észleli, amely akkor is jelez, ha a kapu teljesen lecsukódott, vagy teljesen kinyílt.

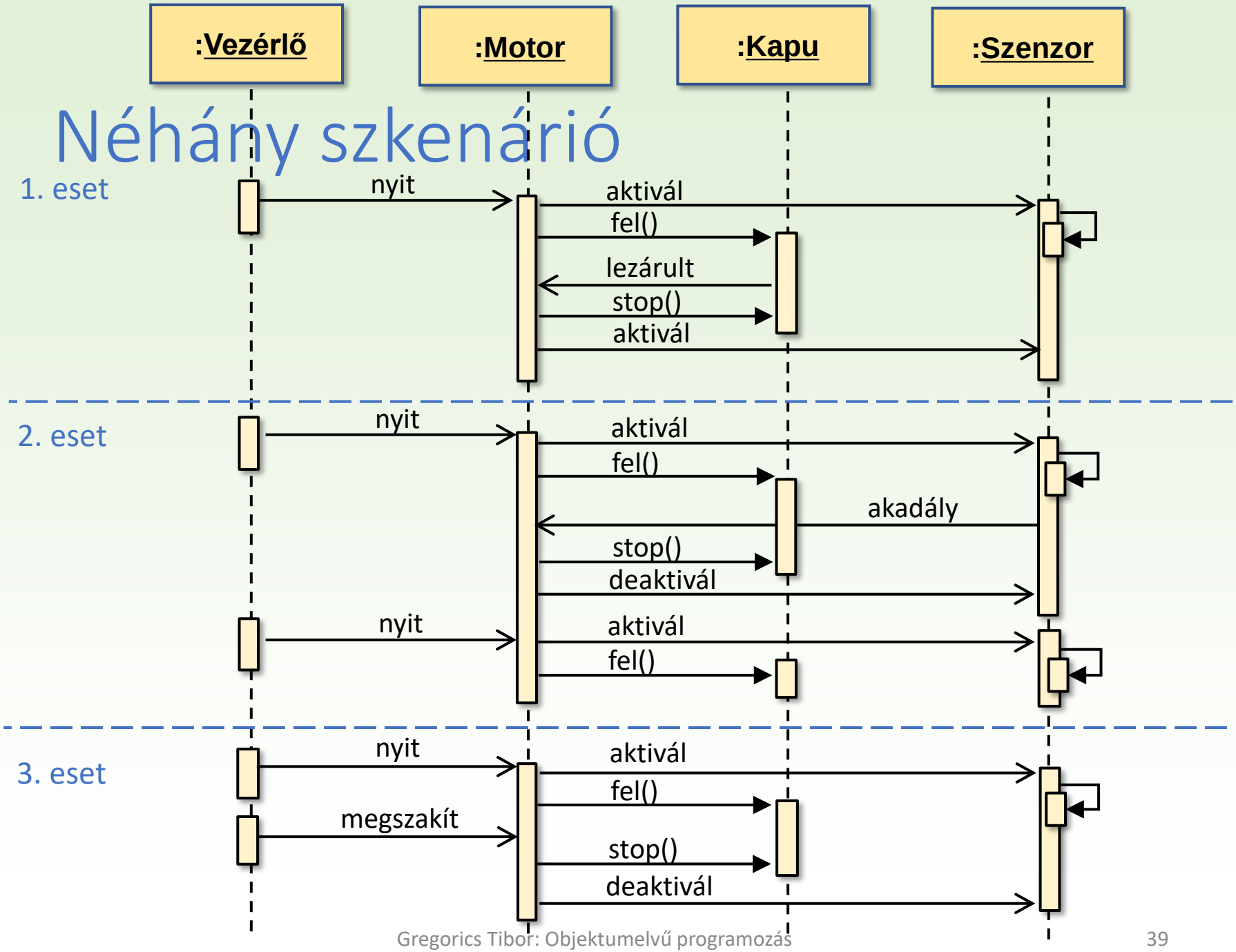
Használati eset diagram



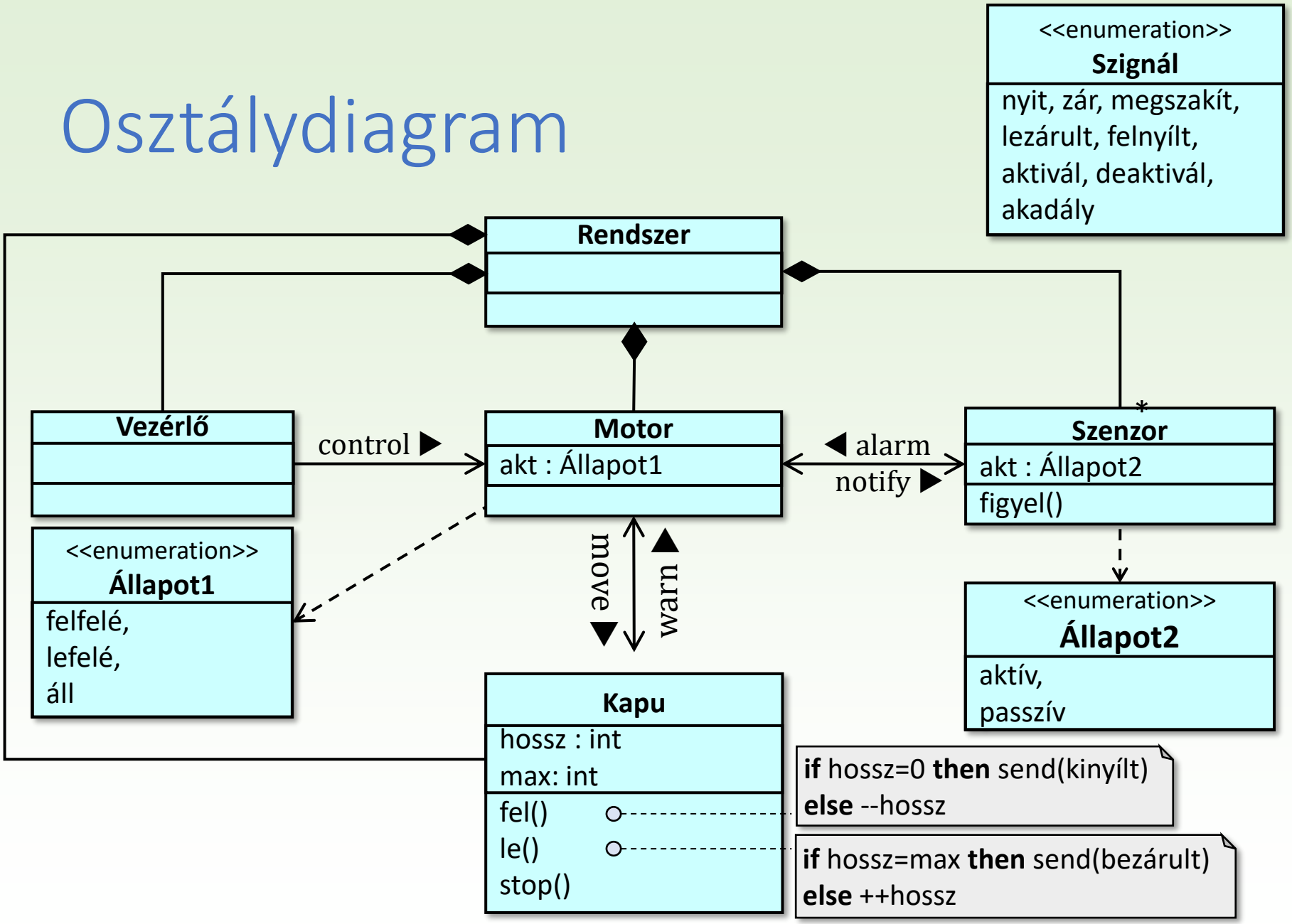
Együttműködési diagram



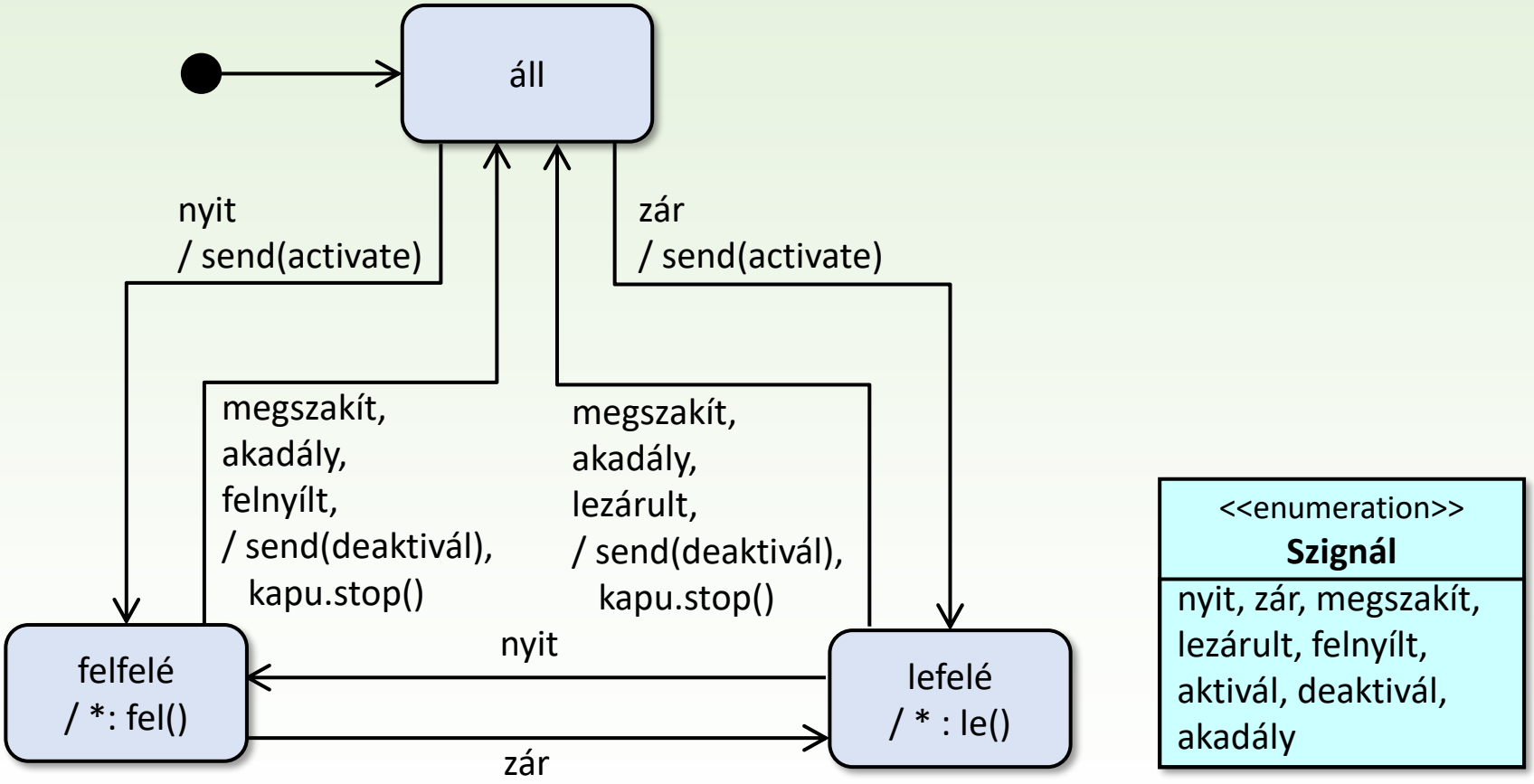
Néhány szkenárió



Osztálydiagram

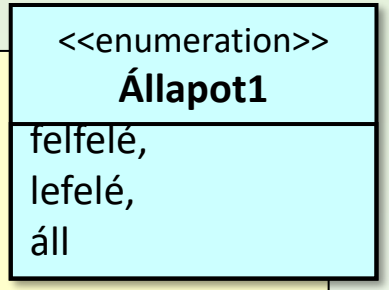


Engine állapotgép diagramja

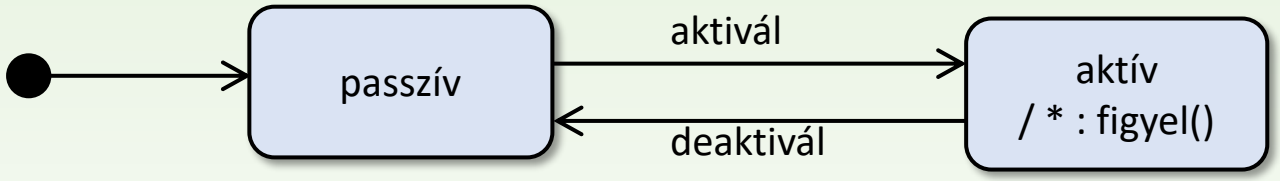


Engine állapot-átmenet kódja

```
switch (akt)
      case áll:
            switch (szignál)
                  case nyit: akt := felfelé; send(aktivál);
                  case zár : akt := lefelé;  send(aktivál);
             end
      case lefelé:
            switch (szignál)
                  case megszakít, akadály, lezárult:
                              akt := áll; kapu.stop(); send(deaktivál);
                  case nyit: currentState = felfelé;
            end
      case felfelé:
            switch (szignál)
                  case megszakít, akadály, felnyílt:
                              akt := áll; kapu.stop(); send(deaktivál);
                  case zár: akt := lefelé;
            end
end
if      (akt = felfelé)  then  kapu.fel()
elseif (akt = lefelé)  then  kapu.le()
endif
```



Senzor állapotgép diagramja

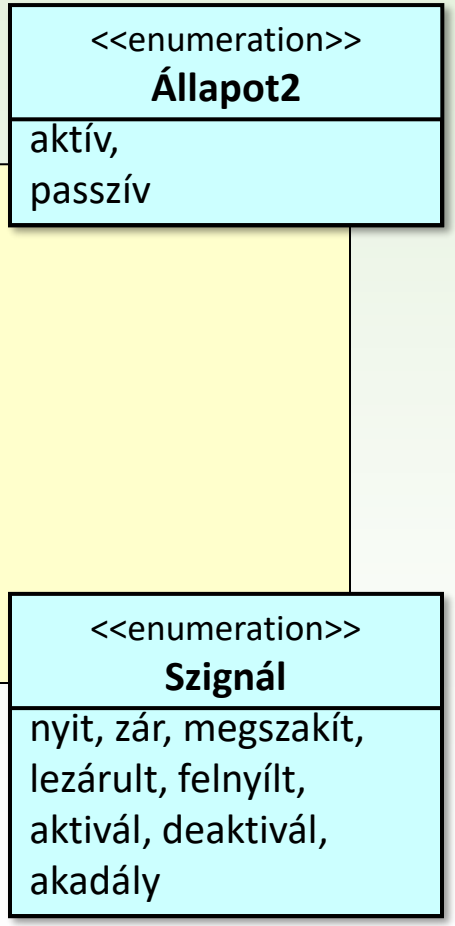


<<enumeration>>
Szignál

nyit, zár, megszakít, lezárult, felnyílt, aktivál, deaktivál, akadály

Senzor állapot-átmenet kódja

```
switch (akt)
        case passzív:
        switch (szignál) {
                case aktivál: akt := aktív;
        end
        case aktív:
                switch (szignál) {
                        case deaktivál: akt := passzív;
                end
end
if (akt = aktív) then  figyel() endif
```



Megvalósítás

- Külön szálakon fut majd az :Engine és a :Senzor. Mindkettőnek saját megvalósítandó állapotgépe van.
  - Habár a :Gate is változtatja az állapotát, de ezt közvetlenül, szinkron hívásokkal idézi elő az :Engine, ezért nem kell külön állapotgépet megvalósítani, metódusai az :Engine állapotgépével közös szálon futnak.
- Aszinkron üzenetek kezeléséhez mind az :Engine, mind a :Senzor egy-egy saját eseménysorban gyűjti a hozzá érkezett üzeneteket, amelyhez mindkettő egy szinkron módon hívható send() metódust biztosít.
  - Így a send() metódus a küldő objektum szálján futva helyez el egy eseményt, az esemény kivétele pedig a fogadó objektum állapotgépének szálán történik.