Szoftvertechnológia és – technikák
5. gyakorlat – állapotgép és aktivitás diagram

1. A gyakorlat menete

A gyakorlat első felében először egy bemelegítő feladaton keresztül rajzolunk egy aktivitásdiagramot. Továbbá egy vezetett feladat keretében megtervezünk és elkészítünk egy állapotgépet, majd elemezzük a hozzá készült kódot. A gyakorlat második felében ezt az állapotgépet és kódot kell kiegészíteni további funkciókkal, önállóan dolgozva.

2. Vezetett feladatok

On-line webshop – aktivitás diagram készítése

FELADAT: Modellezzük a következő on-line webshop működését aktivitásdiagram segítségével!

A vásárló az oldalon található keresés funkciót használva kereshet a termékek között. Ezután, ha megtalálta, amit keresett, részletesen is megtekintheti a kívánt terméket. Amennyiben a termék túl drága a vásárló számára, dönthet úgy, hogy folytatja a vásárlást, más terméket keresve, vagy pedig azonnal be is fejezheti a vásárlást. Amennyiben a termék ára megfelelő a vásárló számára, a vásárló a terméket a kosarába teszi, ezután pedig tovább folytatja a vásárlást, vagy pedig befejezi azt, és checkoutol. Az oldalon checkoutolni bármikor lehet, például akár keresés közben is. Checkout után az oldal megkéri a vásárlót, hogy fizessen, valamint egy felugró ablakban azt is megkérdezi tőle, hogy szeretne-e feliratkozni a hírlevélre. A fizetés folyamatát nem kell részletesen kifejteni. Amennyiben a felhasználó fizetett, és jelezte is a hírlevélre való fel(nem)iratkozási igényét, a folyamatot befejezettnek tekintjük.

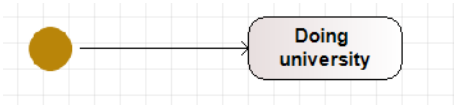
Egyetemi követelményrendszer – állapotgép készítése

FELADAT: Modellezzük egy hallgató egyetemi haladását állapotgép segítségével, az alábbiak szerint!

A hallgató, miután elkezdte az egyetemet, kiválaszthat egy aktív félévet, és elkezdheti az adott félév tárgyait csinálni. A hallgató továbbá kérvényezheti diplomája kiadását, amit, amennyiben a hallgató minden tárgyat elvégzett, meg is kap. Aktív félév elkezdésekor a NEPTUN rendszer regisztrálja a hallgatót az adott félévre, illetve az abban a félévben elvégzendő, még nem teljesített tárgyakra is. A félév befejezésekor a NEPTUN leiratkoztatja a hallgatót a félévről és a tárgyakról, illetve regisztrálja az elvégzett tárgyakat. Jelenleg egy félév van az egyetemen. Ebben a félévben a hallgatónak két tárgyat kell elvégeznie, ezeket természetesen párhuzamosan végzi. Az első tárgynál 2 db ZH-t kell teljesíteni, mindegyik külön-külön egyszer pótolható. A második tárgynál 1 db ZH-t kell teljesíteni, ez nem pótolható. Mindkét tárgy vizsgás; vizsgát a hallgató csak akkor tehet, ha a félévközi követelményeket teljesítette, vagyis van aláírása. A vizsgát csak egyszer lehet egy félévben megpróbálni, pótlási lehetőség nincs. Amennyiben a hallgató a vizsgát is teljesítette az adott tárgyból, a tárgy számára befejezettnek minősül a félév végén. Ha egy tárgyat a hallgató nem teljesített az adott félévben, később, mikor újra megpróbálja a félévet, teljesítheti azt; ekkor, amennyiben volt aláírása, az aláírás megmarad. Ha viszont nem volt aláírása, de a félévközi követelményeket részben teljesítette, a részben teljesített követelmények elvesznek.

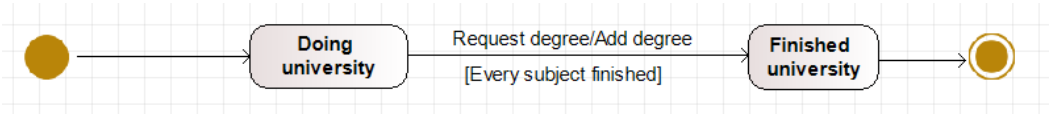
MEGOLDÁS: Hozzunk létre egy új *modelio* projektet, és adjunk hozzá egy új *State Machine diagramot*. A feladat megoldása során érdemes *Model* vagy *Diagram perspective*-be kapcsolnunk.

Adjuk hozzá a kezdőállapotot (*Initial State* a palettán), ahonnan mutasson egy állapotátmenet (*Transition*) a szintén újonnan felvett *Doing university* állapotba (*State*). Ez lesz a hallgató kezdőállapota, ebben az állapotban dönthet úgy, hogy indít egy új aktív félévet, vagy pedig kérvényezi a diplomáját. Kezdjük az utóbbival, mivel ez lesz a rövidebb ág!



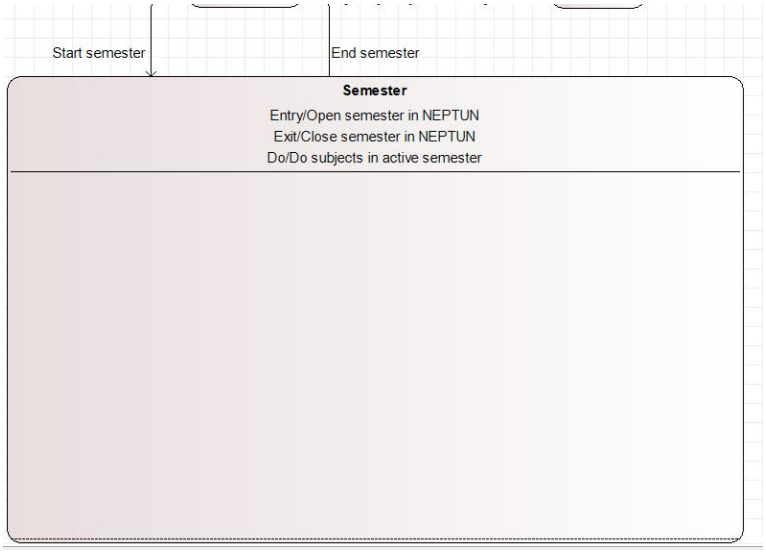
Vegyünk fel egy új állapotot (*Finished university*), amibe a hallgató a diplomája megszerzése után fog lépni. Mutasson ebből az állapotból egy él (állapotátmenet) a végállapotba (*Final State* a palettán), hiszen a diplomája megszerzése után az állapotgép működése befejeződik. A *Doing University* állapotból szintén vegyünk fel egy állapotátmenetet a *Finished university* állapot felé. Az eddigiekkel ellentétben itt viszont még be kell állítanunk néhány dolgot, hiszen a diploma kérvényezése az az esemény, ami kiváltja ezt az átmenetet. Kattintsunk duplán az állapotátmenten (vagy lent kattintsunk a *Properties* fülre), és töltsük ki az adatokat az alábbi ábrának megfelelően. A *Received event* a kiváltó eseményt (előadáson **esemény**), az *Expression of the action* a bekövetkező hatást (előadáson **akció**), a *Guard* pedig az őrfeltételt jelenti (előadáson **őrfeltétel**). Az őrfeltételre azért van szükségünk, mert csak akkor léphet át a hallgató a *Finished university* állapotba, amennyiben minden tárgyat teljesített. Amennyiben a diagramon nem jelennének meg az állapotátmenet feliratai, keressük meg a *Symbol* nyilat (jobb oldalt a képernyő szélén), és pipáljuk be a *Show label* opciót.

Property	Value
Name	Transition
Received event	Request degree
Guard	Every subject finished
Expression of t...	Add degree
Sent signal/ev...	<null>
Post condition	

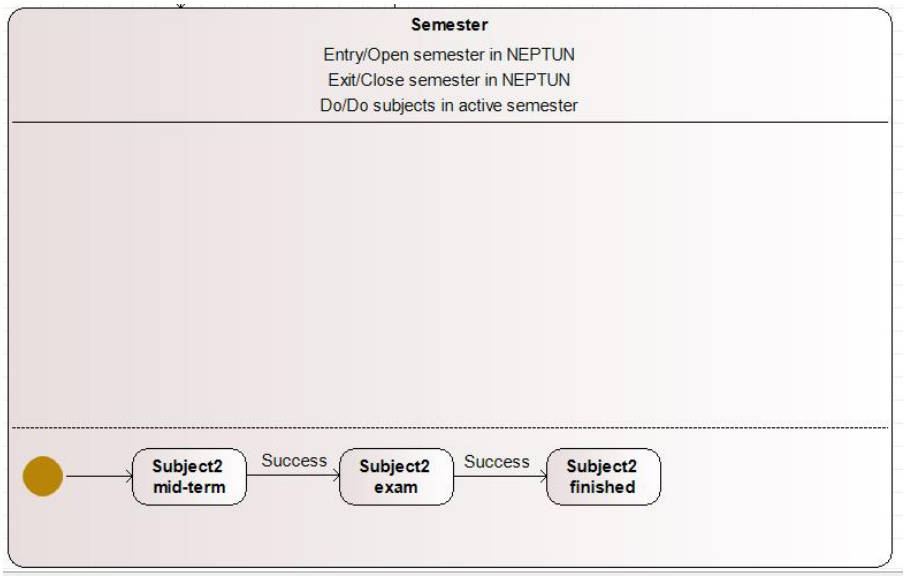


A következő lépés a félév teljesítésének modellezése. Vegyünk fel egy új, *Semester* állapotot, majd vegyük fel a *Doing university* állapotból, illetve az oda vezető állapotátmeneteket (*Start semester* és *End semester* event-ek hatására lépnek át). Adjunk hozzá a Semester állapothoz három *Internal transition-t*: az entry, exit, és do akciókat. Ezek az előadáson elhangzott módon működnek. Töltsük ki az akciókat az alábbiakkal. Ezek a belső akciók fogják a félév indítását, lezárását, és a tárgyakra való regisztrálást jelképezni a NEPTUN rendszerben.

Property	Value
Received event	ENTRY
Guard	
Expression of t...	Open semester in NEPTUN
Sent signal/ev...	<null>
Post condition	



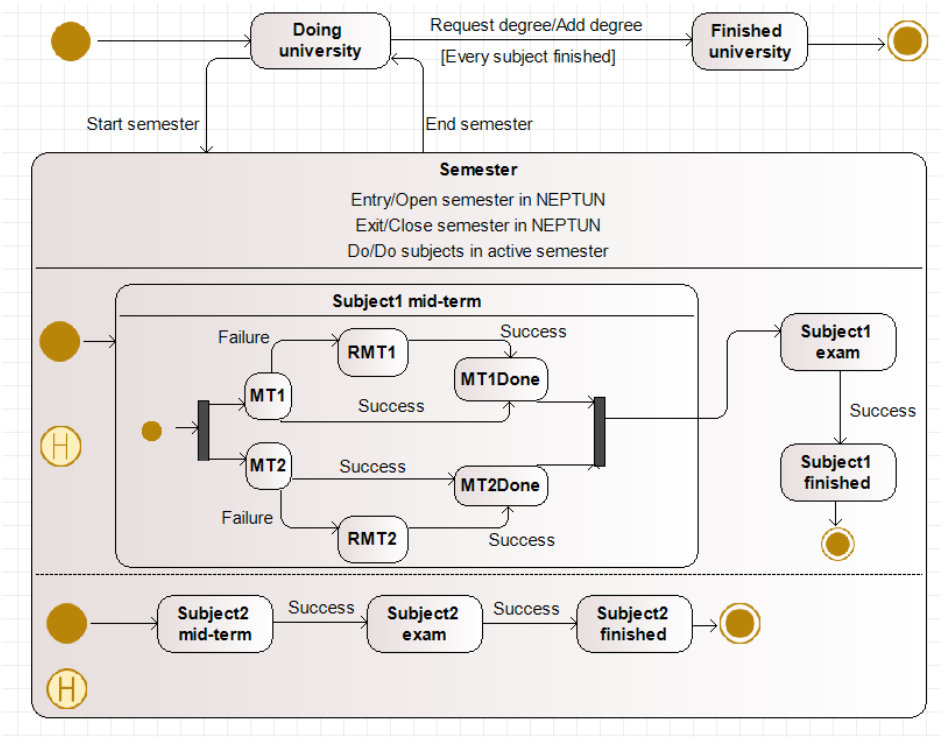
Mivel a féléven belül a hallgató a két tárgyat párhuzamosan végzi, és ezeknek az eredménye független egymástól, ezért egy összetett állapotot (*composite state*) fogunk készíteni. Ehhez vegyünk fel egy *Region*-t a *Semester* állapoton belülre. Kezdjük a második tárgyunkkal, mivel ez egyszerűbb lesz. Először is szükségünk van egy belépési pontra (*Initial State*) a második régión belül, ami egy új állapotba, a *Subject2 mid-term* állapotba mutat egy állapotátmenettel. Vagyis először a hallgatónak a ZH-t kell teljesítenie. Amennyiben ez sikerül (*Success* event-el ellátott állapotátmenet a *Subject2 mid-term* állapotból), a hallgató próbálkozhat a vizsgával (*Subject2 exam* állapot). Majd hogyha ez is sikerül (ismét *Success* event átmenet), akkor a tárgyat sikerrel teljesítette (*Subject2 finished* állapot). Ha bármikor közben elakad a hallgató (pl. nem sikerül a ZH-ja), akkor nem lépünk tovább, az állapotgép elakad. Alternatív megoldásként felvehetnénk egy *Subject2 failed* állapotot, viszont ezt egy kicsit bonyolultabb lenne kezelni később, ezért most nem vesszük fel.



Az első tárgynál hasonlóan gondolkodhatunk. A különbség a félévközi követelményeknél van, ugyanis itt két ZH van, és mindegyik egyszer pótolható. Ezért a kezdőállapot a felső régióban (*Subject1 mid-term*) szintén egy összetett állapot (*composite state*) lesz, amit a *Semester*-hez hasonló módon készíthetünk el.

Mivel mindkét ZH teljesítése szükséges, ezért a kezdőállapotból (*Initial State*) kiindulva használjunk egy *Fork*-ot, majd a végén, a 2 ZH sikerét jelző állapotnál egy *Join*-t (a *Symbol* nyílnál az *Orientation* alatt be tudjuk állítani, hogy függőleges vagy vízszintes legyen). Ha egy ZH nem sikerült, *Failure* átmenettel menjünk a pót ZH-t reprezentáló állapotban. Illetve vegyünk fel egy-egy állapotot a két ZH sikerének is, és ebből induljon a *Join*, hogy a pót ZH eredménye is számítson, de ne kelljen a sima és a pót ZH egyszerre, hiszen ennek nem lenne értelme.

Ami hátramaradt, az a félévközi eredmények elmentése. Erre a problémára jelentenek megoldást a *Shallow History* és *Deep History* indikátorok, az előadáson elhangzottak alapján. Ha *Deep History*-t használnánk, akkor a belső összetett állapotát is elmentenénk, vagyis a részleges félévközi eredmények is megmaradnának. Ez probléma lenne, hiszen például előfordulhatna, hogy a pót ZH-ból indulunk, amit külön kezelnünk kellene. Mivel most ezt nem szeretnénk megtenni (a feladat nem így szólt), ezért *Shallow History*-t használjunk. Ha mindent jól csináltunk, a végén az alábbihoz hasonló állapotgépet kell, hogy kapjunk.



Egyetemi követelményrendszer – forráskód elemzése

FELADAT: Elemezzük a feladathoz készített C# kódot! Vizsgáljuk meg, hogy az egyes állapotok és állapotátmenetek hol jelennek meg a kódban!

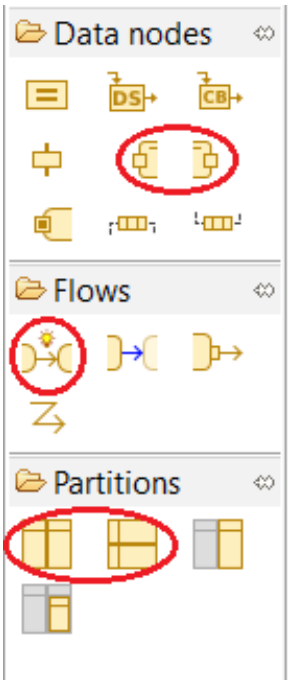
3. Önálló feladatok

Regényírás folyamatának modellezése (2p)

FELADAT: Készítsünk aktivitás diagramot az alábbi specifikáció alapján!

Egy író új regény írására szánja el magát. Először megírja a kéziratot, majd ezután ki is javítja az általa talált hibákat benne. Ezután megmutatja a kéziratot a barátainak, valamint ezzel párhuzamosan postára adja a kézirat másolatát a vele szerződésben álló kiadónak is. Amennyiben a barátoknak nem tetszett a kézirat, az író ismét átnézi és kijavítja a barátok, és az általa talált hibákat is a kéziratban. Ezután újra elküldi a kéziratot a barátoknak és a kiadónak is a korábbiakhoz hasonló módon. A kiadó, miután megkapta a kéziratot, elbírálja azt. Ez a bírálati folyamat maximum 3 hónapig tarthat, ha eddig nem érkezik bírálat, a regényt automatikusan elutasítja a kiadó. A bírálat elkészültekkor, amennyiben az tetszik a kiadónak, a kiadó egy újságon keresztül gratulál az írónak. A regény elutasításáról nincs visszajelzés az író felé a kiadótól. Az író, hogyha a barátoknak is tetszett a kézirat, amint meglátja a gratulációt az újságban, örömünnepet tart.

***TIPP*:** adatfolyam és partíciók modellezésére modelio-ban használjuk az alábbiakat!



Egyetemi követelményrendszer kiegészítése

FELADAT: Egészítsük ki az egyetemi követelményrendszerünket modellező állapotgépet, illetve a hozzá tartozó forráskódot az alábbiak alapján:

- Egészítsük ki a rendszert úgy, hogy a hallgató a diplomája kérvényezése után, amennyiben minden tárgyat teljesített, záróvizsgát kell, hogy tegyen. Amennyiben a záróvizsga sikeres, a hallgató megkapja a diplomáját, egyébként pedig akárhányszor újra próbálkozhat. (**2p**)
    - ***Tipp***: érdemes megkülönböztetni a záróvizsgára bocsájtható hallgatókat. A *Student* tudhatja magáról, hogy sikerült-e a záróvizsga!
- Oldjuk meg, hogy egy tárgy nem feltétlenül kell, hogy vizsgás legyen. A nem vizsgás tárgyakat a hallgató akkor tudja teljesíteni, ha a félévközi követelményeket teljesítette. (**1p**)
- Vezessük be a második félévet (*Semester2*), amit a hallgató csak akkor tud elkezdeni, ha az első félév tárgyai közül mindet teljesítette, vagyis teljesítette az első félévet. A második félév szintén két tárgyat tartalmaz (*Subject3* és *Subject4*), mindkettő vizsgás, és mindkettő pontosan egy darab pótolható ZH teljesítését követeli meg. (**2p**)
    - ***Tipp***: használhatjuk az *IsStudentFinishedSemester* segédfüggvényt a *NEPTUNSystem* osztályban.
- Oldjuk meg, hogy egy vizsgát maximum 3-szor lehessen ismételni a jelenlegi 0 helyett. (**1p**)
- Adjunk támogatást akkreditált tárgyak kezelésére. A hallgató a félév kezdésekor maximum 1 tárgyak akkreditáltathat korábbi tanulmányai alapján, ekkor a tárgy teljesítettnek számít. Akkreditáltatni csak nem teljesített tárgyat lehet. (**2p**)
    - ***Tipp***: A megoldás részeként a *NEPTUNSystem* osztályban érdemes lehet egy extra paramétert felvenni hozzá, és kiegészíteni a *StartActiveSemester* függvényt.

4. Gyakorló feladatok otthonra

Nyomtató – állapotgép készítése

Modellezzük egy, az alábbi módon működő nyomtató működését állapotgép segítségével!

A nyomtató kezdetben ki van kapcsolva, a bekapcsoló gombbal bekapcsolhatjuk. Miután bekapcsolt, lehetőségünk van egy aktív dokumentum kiválasztására, amit aztán a nyomtató oldalanként kinyomtat. Nyomtatás során a papír bekerül a gépbe, a szöveg rányomtatódik, majd a papír kikerül a gépből. Amennyiben kétoldalas nyomtatást választottunk, a nyomtató egymás után, a papír mindkét oldalára nyomtat. A nyomtatás véget ér, amint elfogytak a nyomtatandó oldalak. Ekkor vagy új nyomtatást indítunk (új aktív dokumentum kiválasztásával), vagy pedig kikapcsoljuk a nyomtatót.

Bankautomata – állapotgép elemzése

Készítsünk forráskódot az alábbi állapotgép alapján! A nem kibontott összetett állapotokat (*Customer Authentication* és *Transaction*) nem kell részletezni! *(forrás: https://www.uml-diagrams.org/bank-atm-uml-state-machine-diagram-example.html?context=stm-examples)*