Szoftvertervezés és modellezés UML alapokon

Süle Zoltán, Tarczali Tünde

Közreműködő: Simon Gyula

2014

A tananyag a TÁMOP-4.1.2.A/1-11/1-2011-0088 projekt keretében a Pannon Egyetem és a Miskolci Egyetem oktatói által készült.

H-8200 Veszprém, Egyetem u. 10.

H-8201 Veszprém, Pf. 158.

Telefon: (+36 88) 624-911

Tartalomjegyzék

1. Interjú a megrendelővel: kérdések és válaszok

A tananyag célja, hogy útmutatást adjon a Szoftvertechnológia tárgy gyakorlati óráin résztvevő hallgatóknak. A feladat egy rendszer tervének elkészítése a felméréstől a rendszer tervezéséig, majd ez alapján a rendszer fejlesztése. Ennek megfelelően a tananyagban követjük a szoftverfejlesztés elméleti órán megismert fázisait. Első lépésben történik meg a feladat megismerése, valamint a rendszer funkcionalitásainak felmérése.

Mivel a fejlesztés és a felmérés során iteratív módszer követése javasolt, ezért a felmérések és a modellalkotási fázisok több lépcsősek. A modellezés során szinte kivétel nélkül előfordul, hogy felmerülnek olyan kérdések, amelyekkel vissza kell térni a megrendelőhöz, és pontosítani kell a részleteket. Ezek a részletek vonatkozhatnak a funkcionális követelményekre, de akár nemfunkcionális követelményekre is, mint például a rendszer válaszidejére, rendelkezésre állására, vagy a felhasználói felület részleteire.

Miután a megrendelő úgy döntött, hogy egy új rendszer fejlesztésére vagy egy régebbi rendszer kiegészítésére ad le megrendelést, célunk ennek a rendszernek a felmérése. Fel kell készülni arra, hogy a megrendelő a saját szavaival, úgynevezett szaknyelven fogja meghatározni, hogy milyen terület fejlesztéséhez van szüksége az új szoftver elkészítésére. Az informatikus feladata ilyenkor kettős: meg kell ismerkednie a szaknyelvi fogalmakkal, valamint azonosítani kell egy beszélgetés, úgynevezett üzleti interjú alapján melyek azok a funkciók, amelyekkel a rendszernek rendelkeznie kell. A szakterület szavait érdemes szakterületi szótárba jegyezni a használt szinonimákkal együtt, hiszen a fejlesztett rendszer is ezen a „nyelven” kell szóljon a felhasználóhoz. A szakterületi szótár elkészítése nem a jelen tananyag tárgya. A következő fejezetben látunk példát egy elkészített üzleti interjúra.

Menetrend készítő alkalmazás fejlesztése busztársaság számára

Az üzleti interjú lefolytatása a megrendelővel általában egy hozzáértő informatikus vagy menedzser feladata. Optimális esetben mindketten részt vesznek a beszélgetésben. Ez azért is fontos, mivel több szem többet lát alapon más-más nézőpontból látják a megvalósításra kerülő projektet. Az üzleti interjú tehát a megrendelő és a fejlesztéssel megbízott cég, vállalkozás képviselői között a megvalósításra kerülő rendszer funkcionális és egyéb követelményeinek rögzítésére megrendezett tárgyalás.

Az elkészült tananyagban egy busztársaság számára fejlesztendő menetrend készítő alkalmazás modellezését mutatjuk be. Első körben tehát elkészült az üzleti interjú egy nemformális leírása, amely az alábbiakban olvasható:

Menetrend készítő alkalmazás üzleti interjúja

Az alkalmazás feladata, hogy segítséget nyújtson Széplak város busztársaságának a menetrend elkészítésében. A menetrendet általában a diszpécserek állítják össze. A menetrendben a diszpécser az egyes útvonalakat és a társaság tulajdonában álló autóbuszokat rendeli össze a társaság alkalmazásában álló sofőrökkel. A sofőröknek a rendszerbe való belépéskor látniuk kell a heti munkabeosztásukat, tehát hogy melyik napon melyik buszjárattal kell közlekedniük, és milyen időtartamban. A rendszer figyelje azt is, hogy egy sofőr egy napon napi 8 óránál többet nem dolgozhat, és ebben 4 és fél óra után 45 percnyi pihenőidőnek kell lennie. A buszok reggel 4:00-tól közlekednek éjjel 11:30-ig, de az útvonalakon a pontos menetrendet a diszpécser adja meg. Ahogyan azt is, hogy az egyes útvonalak milyen megállókat tartalmaznak, és mennyi idő kell a busznak a megállóhoz történő érkezéshez a kiindulási megállótól számítva.

A buszok és a sofőrök adatait a rendszerben a titkár kezeli. Az ő feladata az újonnan felvett sofőrök és beállított buszok felvitele is. Ezen kívül természetesen szerkesztheti is ezeket az adatokat. A sofőrök a saját adataik nagy részéhez hozzáféréssel rendelkeznek, és azokat módosíthatják is. A rendszernek tudnia kell kezelni azt is, hogy ha egy sofőr beteg lesz, és helyettesítésére van szükség.

A buszok állapotában történt változásokat a buszhoz rendelt sofőr vagy a titkár jelezheti a rendszerben. Ennek alapján a szervizben dolgozó szerelő munkatárs lekérdezheti a szerelésre váró buszokat és a javítás befejezése után a szervizkönyvbe kell rögzítenie az elvégezett munkát.

A fent leírt üzleti interjú egy elsőkörös interjúnak tekinthető. Mint látható nem közöl minden egyes funkciót részletesen, de ezt nem is várhatjuk el egy informatikai rendszerhez nem értő megrendelőtől. A modellezésben az informatikus feladata a hiányosságok megtalálása, és ezek alapján további követelmények meghatározása a megrendelővel együttműködve. Következőkben feladatunk a rendszer szereplőinek és a hozzájuk kapcsolódó funkcióknak a meghatározása, amelyet a használati eset diagramokkal modellezünk.

2. A rendszer követelményeinek meghatározása lépésenként: use case diagramok, use case táblázatok

A rendszer működésének határai a használati esetek felsorolásával meghatározhatók. Emellett meghatározható hogy az egyes aktorok mely folyamatok részei lesznek. Az aktorok nem mások, mint a rendszerrel kapcsolatban álló személyek vagy dolgok (rendszerek, adatbázisok), amelyek nem képezik a rendszer szerves részét, de a folyamatokra hatással vannak. A használati esetek meghatározásával lefektetésre kerülnek a rendszerrel szemben támasztott követelmények. Fontos, hogy a rendszer tervezése során hangsúlyt fektessünk ezen esetek pontos meghatározására együttműködésben a rendszer megrendelőivel illetve felhasználóival, hiszen a pontos tervezés a szoftverfejlesztés későbbi lépéseinél idő és költségmegtakarítás szintjén jelentkezhet.

Modellezés alapjai

A modellezés első lépésében a feladat a szakterület és a létrehozandó szoftver megismerése, a funkcionalitások azonosítása. Ennek során a szakterület képviselőivel úgynevezett üzleti interjúkat kell készíteni. A felmérés után az interjú lejegyzésével egy szöveges leírást kapunk, amelyet táblázatos formában, strukturált szövegként kell feldolgozni.

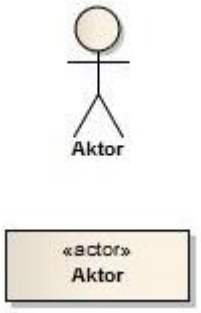
Ez alapján történik meg a rendszer funkcionális követelményeinek meghatározása, azonosítása, amelynek első lépésében az üzleti folyamatok, illetve használati esetek meghatározására a feladat.

A rendszerek egy cél szolgálatában jönnek létre. Ezek a célok funkcionális és nemfunkcionális követelményekként azonosíthatók. A funkcionális követelmény nem más, mint a tulajdonképpeni feladat, amit a rendszernek el kell végeznie. A nemfunkcionális követelmény az egyéb megszorításokat takarja, mint például mennyi ideig futhat egy kérés a rendszerben, mennyi idő áll rendelkezésre a válaszadásra.

A rendszer aktorainak azonosítása

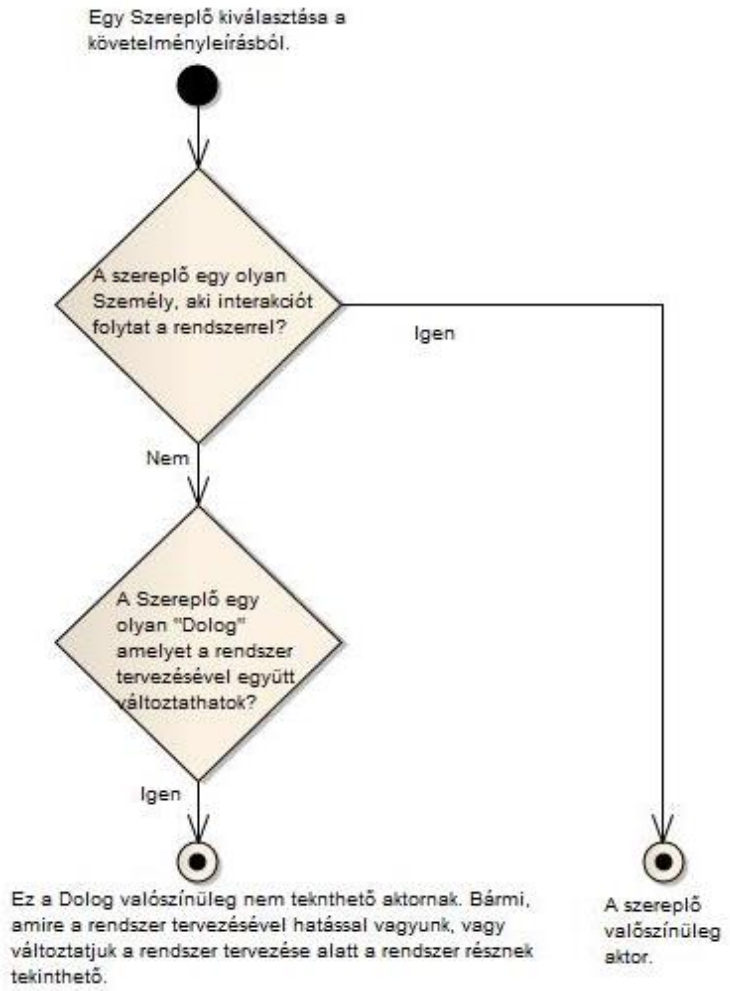
A rendszer funkcionalitásának meghatározásánál az első lépés a rendszerrel kapcsolatban lévő szereplők, az aktorok meghatározása, valamint a használati esetek azonosítása. Aktor nem csak személy lehet, hanem bármely a rendszerrel kapcsolatban lévő rendszer is.

Az aktorok jelölésére a szabvány a következő ábrán látható módszereket használja. A felső ábrán a leggyakrabban használt úgynevezett pálcikaemberes jelölés alatt pedig egy másik lehetőség, sztereotípiával jelölt aktor.



1. ábra: A rendszerrel kapcsoltban álló aktorok jelölési lehetőségei az UML nyelvben.

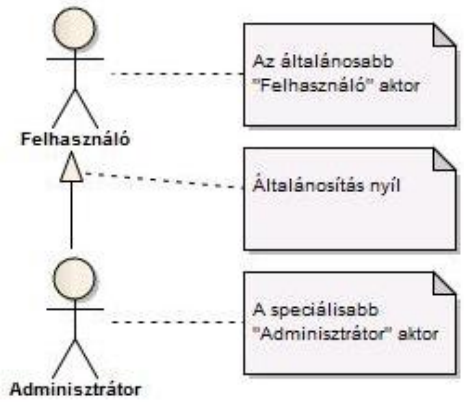
A következő folyamatábra lépései segítséget nyújtanak ahhoz, hogy eldönthessük, egy személy vagy tárgy aktorként tekinthető-e a rendszerben.



2. ábra: Aktorok azonosítása

Figyelnünk kell arra, hogy néhány esetben Személy szereplő is lehet a rendszer része.

Az aktorok azonosításának folyamán az is kiderülhet, hogy néhány aktor kapcsolatban van egymással. Ezen aktorok mindegyike természetesen a rendszerrel interakcióban van. Az aktorok közötti kapcsolat modellezésére mutat példát az alábbi ábra.



3. ábra: Felhasználó és adminisztrátor aktorok közötti specializációs kapcsolat

A felhasználó és az adminisztrátor aktorok között specializációs kapcsolat áll fenn. Ez azt jelenti, hogy az adminisztrátor a rendszer egy speciális felhasználója. Minden olyan funkcióhoz hozzáférése van, amihez a felhasználónak, ezen kívül azonban rendelkezik más, hozzáadott funkciókkal. A specializációs kapcsolat a másik oldalról tekintve általánosítást jelent.

A rendszer használati eseteinek azonosítása

Az aktorok egészének vagy egy részének azonosítása után a következő lépés az interakciók azonosítása. Ezek az interakciók a rendszer követelményei alapján azonosíthatóak. A korábban leírt üzleti interjú alapján azonosítani kell a konkrét követelményeket. A használati esetek feladata ezen követelmények egyértelmű azonosítása és leírása. Ezen leírás alapján egyértelműen meghatározásra kerül, hogy melyek azok a funkciók, amelyeket a rendszernek meg kell valósítania. Ezek a leírások lesznek segítségére a megrendelőnek, a tesztelőnek, és a fejlesztőnek, hogy bizonyosan azok a funkciók kerüljenek megvalósításra és olyan céllal valamint eredménnyel, amelyre szükség van.

Egy használati eset komplexitása az egészen egyszerűtől, mint például egy belépés funkció, az egészen komplexig, mint például több rendszerből történő adatbegyűjtésig terjedhet. Azonban minden használati eset tartalmaz interakciókat a rendszer és az aktor között, valamit kimenete illetve eredménye is van a használati esetnek. A használati esetek szabvány szerinti jelölése látható az alábbi ábrán.

4. ábra: Használati eset jelölése az UML szabványban

Ahogyan az ábrán látható a használati esetek jelölésére az ellipsziseket alkalmazzák. Az ellipszisbe kerül beírásra a használati eset neve. A használati esetek elnevezésén figyelembe kell venni, hogy lehetőség

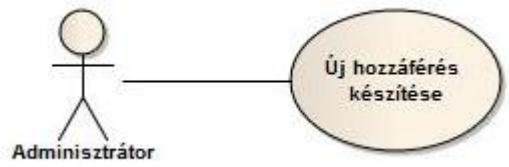
> ***1. követelmény:*** *a vállalati Közösségi oldalhoz való hozzáférést egy Adminisztrátor felügyeli. A hozzáférés készítését az Adminisztrátor kezdeményezi a rendszerben a beérkezett kérések alapján.*

szerint tömören, de egyértelműen azonosítsa a használati eset által lefedett funkciót.

A használati esetek tehát a követelmények alapján kerülnek meghatározásra. Az ábrán látható használati eset egy zárt közösségi oldalhoz való hozzáférés készítésének funkcióját azonosítja. Ennek követelménye a következő:

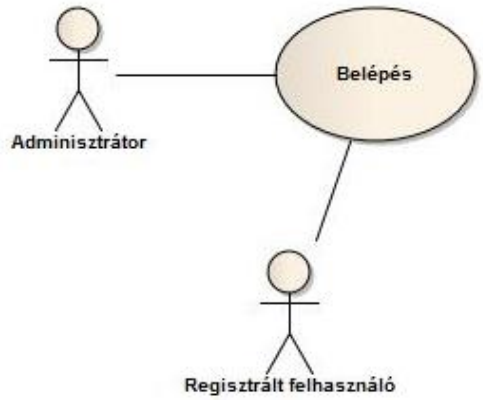
Kapcsolatok, kommunikációs vonalak

A használati esetek és az aktorok közötti kapcsolatokat a kommunikációs vonalak mutatják meg. A jelölés azt az információt hordozza, hogy az aktor részt vesz a használati esetben. Ilyen kapcsolatot mutat az alábbi ábra.



5. ábra: A kommunikációs vonal mutatja, hogy az aktor részt vesz a használati eset lefolyásában

Az ábrán látható kapcsolat írja le azt a követelményt, miszerint az Adminisztrátor rendelkezik egy olyan funkcióval, amely célja egy új hozzáférés elkészítése készítése a rendszerben. Az ábrán egy aktor és egy használati eset kapcsolata van jelölve. Több aktor és több használati eset kapcsolódására is lehetőség van. Erre mutat példát a következő ábra.

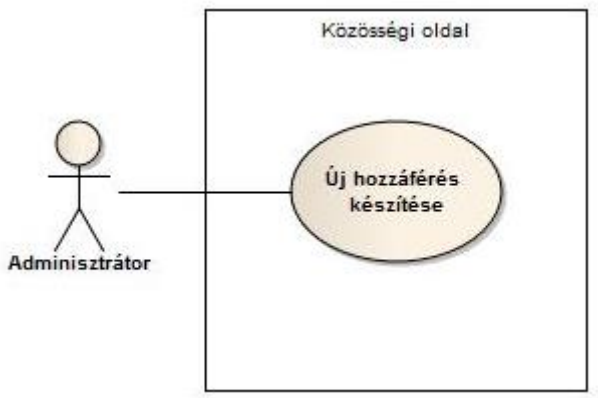


6. ábra: Egy használati esethez több aktor kapcsolódhat

Értelemszerűen, a rendszerbe több felhasználó beléphet, így az Adminisztrátor és a Regisztrált felhasználó is. Ennek megfelelően a Belépés használati esethez mindkét felhasználó hozzá van kötve kommunikációs vonallal. A kommunikációs vonal azt mutatja, hogy a felhasználó részt vesz a használati esetben. Ilyenkor általában többirányú az információáramlás, és az UML ebben az esetben nem tér ki arra, hogy az információ melyik féltől melyikhez jut el, illetve ki indította a kapcsolatot. Ezen kérdések azonosítására lehetőség van a használati eset táblázatokban.

A rendszer határainak meghatározása

A használati eset diagram egyértelműen megadja az aktorok és a rendszer elkülönítését. Az UML lehetőséget ad arra, hogy ezt teljesen tiszta módon legyen jelölhető. A használati eset diagramon a rendszer határait egy keret azonosítja. A keret felső része tartalmazz a rendszer elnevezését.



7. ábra: A rendszer határainak azonosítása a Vállalat Közösségi oldalt megvalósító rendszer esetében

Használati eset táblázatok

A használati eset diagram segítséget nyújt a rendszer funkcióinak és aktorainak azonosításában, de nem elég részletes ahhoz, hogy minden eshetőséget leírjon. A használati eseteket ezért használati eset

táblázatokkal írjuk le. A táblázatok segítséget nyújtanak a rendszer tervezőinek többek között abban, hogy megérthessék, egy adott használati esetnél ki a legfontosabb aktor vagy mely lépésekből áll a használati eset. A használati esetek azonosítását és részletes leírását segítik a következő lépések:

1. Azonosítsuk, ki fogja használni a rendszert.
2. Válasszunk ki egy aktort.
3. Azonosítsuk, hogy az aktor ilyen interakciókat végez a rendszerrel. Minden ilyen interakció használati esetnek tekinthető.
4. Minden használati esethez határozzuk meg történő eseményeket.
5. Írjuk le a használati eset leírásban az általános esetben (normál lefutás) bekövetkező eseményeket.
6. Határozzuk meg lépésenként, hogyan viselkedik az aktor és mi erre a rendszer válasza. Határozzuk meg, mire kell az aktornak figyelnie a működéssel kapcsolatban.
7. A normál lefutás azonosítása után határozzuk meg a lehetséges alternatív eseményeket, amelyek a használati eset kibővítései lesznek.
8. Keressünk azonosságokat a használati esetek lefutásában. Vegyük ki ezeket a részeket és általános használati esteként azonosítsuk őket.
9. Ismételjük a 2-8 lépéseket minden aktorra.

Ezeket a leírásokat egységes formára kell hozni, hogy bizonyos minőségi kritériumoknak megfeleljenek. Ez az egységes forma egy táblázat lehet. A minőségi kritériumok a következők:

- Érthetőség: ennek biztosításával egyszerűbbé válik a folyamat lényegének gyors és hibátlan megértése
- Teljesség: egyetlen fontos tényező fölött sem siklik el a figyelem
- Összehasonlíthatóság: a különféle folyamatok összehasonlíthatóak lesznek tartalmilag, valamint az absztrakciós szintet és a folyamatleírások minőségét is tekintve

Az itt látható táblázatos forma egy lehetőség az üzleti folyamatok szabványos leírására. Az irodalomban ez többféle módon megjelenhet, a főbb jellemzői viszont azonosak az egyes táblázattípusoknak.

Használati eset neve	A használati eset leírás részeinek célja és hasznossága
Követelmény	Azok a követelmények, amelyeket a használati eset teljesít
A használati eset célja	A használati eset helyezte a rendszerben és az eset hasznossága
Előfeltételek	Minek kell bekövetkeznie ahhoz, hogy a használati eset lefuthasson
Sikeres lefutás feltétele	A rendszer körülményei a használati eset sikeres lefutásának következtében
Sikertelen lefutás feltétele	A rendszer körülményei a használati eset sikertelen lefutásának következtében
Elsődleges aktor	A használati eset fő aktora. Gyakran az az aktor, aki a használati eset lefutását elindítja, vagy aki direkt információt kap a használati eset lefutásának következtében

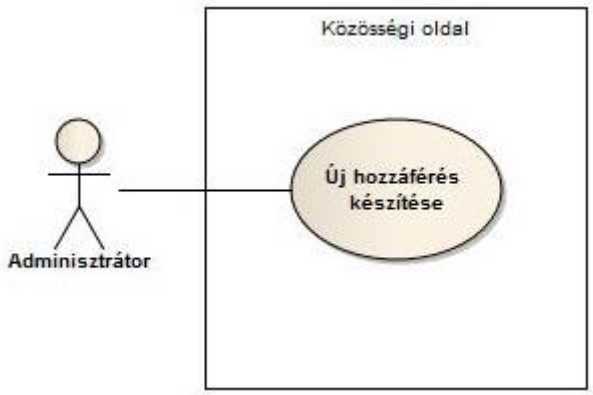
Másodlagos aktor	Olyan aktorok akik részt vesznek ugyan, de nem játszanak fő szerepet a használati esetben
Kiváltó esemény	Egy aktor által elindított esemény, amely a használati eset lefutásához vezet
Fő lépések	Azok a lépések, amelyeken végighaladva a használati eset sikeres lefutásához vezetnek
Kivételek	Olyan alternatív lépések leírása, amelyek a fő lépésektől eltérnek

Az új hozzáférés készítés használati eset táblázata látható a következőkben.

Használati eset leírás részei	**Új hozzáférés készítése**	
Követelmény	1. követelmény	
A használati eset célja	Egy felhasználó hozzáférési igénnyel fordul az Adminisztrátorhoz	
Előfeltételek	A rendszerbe csak korábban igényt leadott felhasználókat lehet regisztrálni, tehát szükséges a felhasználó hozzáférésének elkészítéséhez egy igény megléte	
Sikeres lefutás feltétele	Új hozzáférés készül a felhasználó részére	
Sikertelen lefutás feltétele	Az új hozzáférés elkészítése visszautasításra kerül	
Elsődleges aktor	Adminisztrátor	
Másodlagos aktor	Igények adatbázisa	
Kiváltó esemény	Az adminisztrátor utasítást ad a hozzáférés elkészítésére	
Fő lépések	Lépések	Tevékenységek
	1	Az adminisztrátor utasítást ad a hozzáférés elkészítésére
	2	A rendszer bekéri a felhasználó adatait
	3	Az adminisztrátor megadja a felhasználó adatait.
	4	Az igények adatbázisa visszaigazolja a leadott igényt
	5	Az új hozzáférés elkészül
	6	Az új hozzáférés elkészüléséről a felhasználó emailben értesítést kap.
Kivételek	Lépések	Elágazó tevékenységek
	4.1	Az igények adatbázisa nem igazolja vissza a leadott igényt
	4.2	A hozzáférés elkészítése visszautasításra kerül

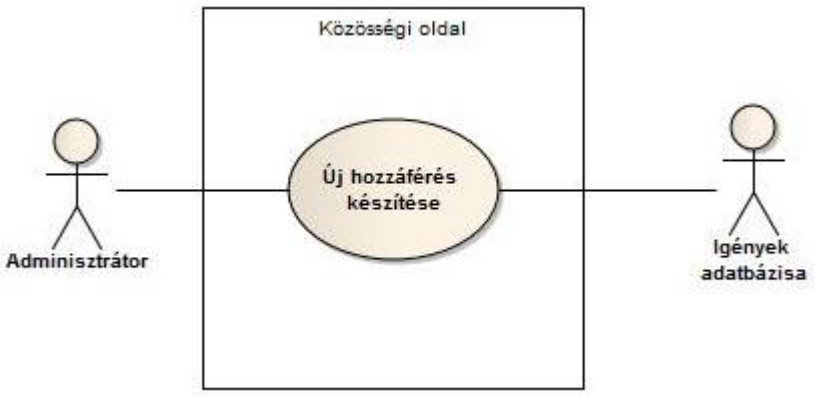
A használati eset táblázat szerves részét képezi a használati esetek meghatározásának. A diagramok önmagukban nem elegendőek a funkciók meghatározásához.

A használati este táblázat nem teljesen egyezik meg a korábban bemutatott használati eset diagramnak. Az új hozzáférés elkészítése a diagramon 1 felhasználóval kapcsolódik.



8. ábra: A használati eset táblázatban és a diagramban lévő információknak konzisztensnek kell lennie

A használati eset táblázatban két aktor kapcsolódik a hozzáférés készítése használati esethez. Ez az aktor az Igények adatbázisa. Minél részletesebben belemerülünk a rendszer funkcióinak a felmérésébe, annál fontosabb, hogy visszatérjünk a korábban elkészített táblázatokhoz vagy diagramokhoz és a leírásokat, jelöléseket konzisztenssé tegyük.



9. ábra: A használati eset diagram frissítése az Igények adatbázisa aktor hozzáadásával

A használati esetek elkészítése során érdemes néhányszor egyeztetni a megrendelővel. Ez azért fontos, mivel a korai fázisban kiderülhet, hogy bizonyos funkciókat, aktorokat nem vettünk számításba, vagy nem merültek fel az addigi egyeztetések során. A későbbiekben jelentős munkával pótolható funkciók korai azonosítása fontos a szoftverfejlesztés költségeinek és időigényének megadott határokon belül tartásához.

A használati esetek száma több tényezőtől függ. Nincsenek egyértelmű szabályok arra, hogy hány use case lesz a rendszerben. A funkciók felmérése és azonosítása során derül az ki, hogy adott rendszer mennyi használati esetet tartalmaz majd. A szám lehet nagyon kicsi, akár 2, de akár több száz is.

Természetesen a rendszer méretétől ez függeni fog. Fontosabb a használati esetek pontos, részletes és körültekintő leírása a tervezés során, és nem az, hogy hány darab használati esettel dolgozunk.

Használati esetek kapcsolatai

A használati esetek írják le, hogy a rendszerben milyen funkciókat kell megvalósítani. A használati esetek leírásakor felmerülhet, hogy egyes esetek kapcsolatban állnak egymással. Észrevehető, hogy vannak köztük hasonlóságok, speciális esetei is előfordulhatnak egy használati esetnek. Lehetnek olyanok, amelyek egy másik esethez valamilyen feltétel lefutásával kapcsolódnak. Ezeket a kapcsolatokat érdemes a használati eset diagramon is megjeleníteni.

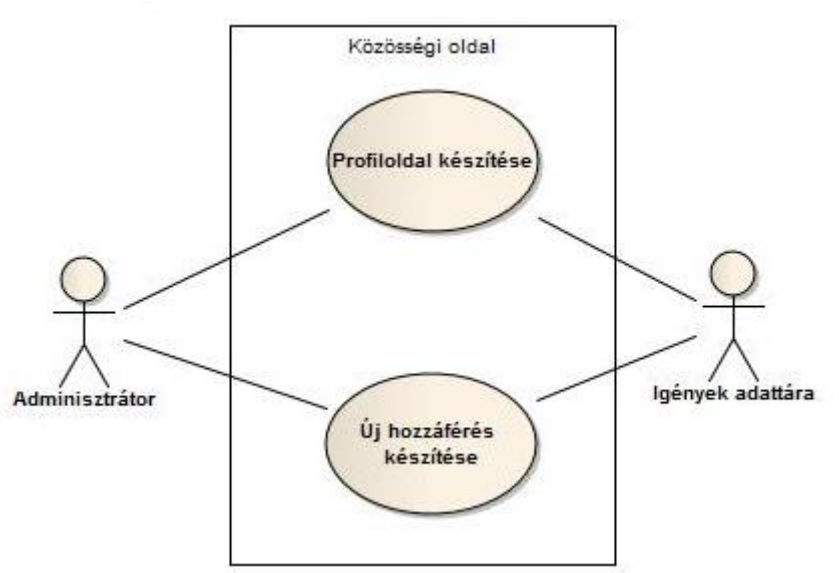
Az <<include>> kapcsolat

A használati esetek, amikkel eddig megismerkedtünk a rendszerrel kapcsolatban álló valamely aktorhoz kapcsolódtak, és céljuk a rendszer funkcióinak leírása volt. A használati esetek közötti kapcsolatok ezen funkciók felosztását célozzák. Ez azért szükséges, hogy a rendszer tervezőit segítve a

> ***2. követelmény:*** *a vállalati Közösségi oldalhoz profiloldal elkészítését egy Adminisztrátor felügyeli. Az Adminisztrátor kezdeményezi a profiloldal hozzáadását a rendszerben a korábban beérkezett kérések alapján.*

rendszerben lévő kapcsolatok olyan szinten meghatározásra kerüljenek, hogy a funkciók megvalósítása minél kevesebb ismétlődést vonjon maga után.

A következő funkció, amelyet a rendszer megvalósít, a profiloldal hozzáadása. Ahhoz hogy a profiloldal elkészüljön, a felhasználónak igényt kell leadnia rá. A funkció használati esetként megjelenik a használati eset diagramon, illetve elkészül a használati eset táblázat, amely részletesen leírja a funkciót. Ez látható az alábbiakban.

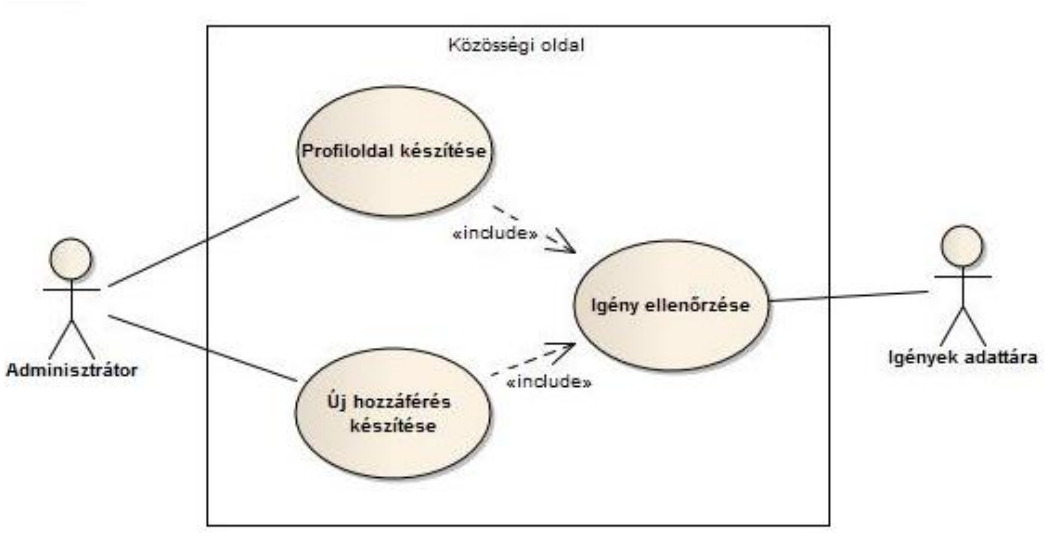


10. ábra: Az új követelmény gyakran új használati esetként jelenik meg a diagramban. Nem minden esetben új használati eset egy új követelmény.

Használati eset leírás részei	Profiloldal készítése	
Követelmény	2. követelmény	
A használati eset célja	Egy felhasználó profiloldal készítési igénnyel áll elő az adminisztrátor felé.	
Előfeltételek	A rendszerben csak korábban igényt leadott felhasználókat rendelkezhetnek profiloldallal, tehát szükséges az elkészítéshez egy igény megléte	
Sikeres lefutás feltétele	Profiloldal készül a felhasználó részére	
Sikertelen lefutás feltétele	A profiloldal elkészítése visszautasításra kerül	
Elsődleges aktor	Adminisztrátor	
Másodlagos aktor	Igények adatbázisa	
Kiváltó esemény	Az adminisztrátor utasítást ad a hozzáférés elkészítésére	
Fő lépések	Lépések	Tevékenységek
	1	Az adminisztrátor utasítást ad a profiloldal elkészítésére
	2	A rendszer bekéri a felhasználó adatait
	3	Az adminisztrátor megadja a felhasználó adatait.
	4	Az igények adatbázisa visszaigazolja a leadott igényt
	5	A profiloldal elkészül
	6	A profiloldal elkészüléséről a felhasználó emailben értesítést kap.
Kivételek	Lépések	Elágazó tevékenységek

	4.1	Az igények adatbázisa nem igazolja vissza a leadott igényt
	4.2	A hozzáférés elkészítése visszautasításra kerül

Észrevehető hogy van olyan lépés a két használati esetnél, amelyek megegyeznek. Az Új hozzáférés készítése és a Profiloldal készítése használati eseteknél ellenőrzésre kerül, hogy a felhasználó nyújtott e be igényt a korábban. Ez a lépés tehát azonos a két használati esetben. Az azonos lépéseket érdemes külön kezelni. Ekkor az azonos lépések külön használati esteként kerülnek megvalósításra és <<include>> kapcsolattal kapcsolódik azokhoz a használati esetekhez, amelyekből származik. Ez látható az alábbi ábrán.



11. ábra: Azonos lépések többszörös használatának kezelésére <<include>> kapcsolat bevezetése.

Az <<include>> kapcsolat meghatározza, hogy az a használati eset, amelyből a nyíl kiindul, teljes egészében felhasználja - tartalmazza – a nyíl által hivatkozott használati esetet. Az előző ábrán látható módon az Új hozzáférés készítése és a Profiloldal készítése használati esetek teljes egészében tartalmazzák az Igény ellenőrzése használati eset minden lépését.

Az ábrán az is új elemként jelenik meg, hogy az Igény ellenőrzése használati eset nem közvetlenül kapcsolódik az Adminisztrátor aktorhoz. Itt az aktor közvetett kapcsolatban áll az Igény ellenőrzésével a többi használati eseten keresztül. Az Igény ellenőrzése használati eset csak az Igények adatbázisához kapcsolódik. Ezzel egyértelműen meghatározható, hogy az Igények adatbázisa egyedül az Igények ellenőrzése használati esettel van kapcsolatban.

Az új kapcsolat leírását át kell vezetni az eddig leírt használati eset táblázatokba is. Ennek megfelelően mind az Új hozzáférés készítése, mind a Profiloldal készítése használati esetek lépései közül a redundáns lépések eltávolításra kerülnek. Ehelyett a Tartalmazott esetek mező, és az include::<használati eset neve> kapcsolattal kerül megjelenítésre a leírásban, amely azt jelöli, hogy minden, a tartalmazott használati esetben lévő minden lépés a tartalmazó használati esetben újrafelhasználásra kerül. Az átalakított használati eset táblázatok láthatóak az alábbiakban.

Használati eset leírás részei	**Új hozzáférés készítése**	
Követelmény	1. követelmény	
A használati eset célja	Egy felhasználó hozzáférési igénnyel fordul az Adminisztrátorhoz	
Előfeltételek	A rendszerbe csak korábban igényt leadott felhasználókat lehet regisztrálni, tehát szükséges a felhasználó hozzáférésének elkészítéséhez egy igény megléte	
Sikeres lefutás feltétele	Új hozzáférés készül a felhasználó részére	
Sikertelen lefutás feltétele	Az új hozzáférés elkészítése visszautasításra kerül	
Elsődleges aktor	Adminisztrátor	
Másodlagos aktor	-	
Kiváltó esemény	Az adminisztrátor utasítást ad a hozzáférés elkészítésére	
Tartalmazott eset	Igény ellenőrzése	
Fő lépések	Lépések	Tevékenységek
	1	Az adminisztrátor utasítást ad a hozzáférés elkészítésére
	2	A rendszer bekéri a felhasználó adatait
	3	Az adminisztrátor megadja a felhasználó adatait.
	4 **include:: Igény ellenőrzése**	A felhasználói igény ellenőrzésre került.
	5	Az új hozzáférés elkészül
	6	Az új hozzáférés elkészüléséről a felhasználó emailben értesítést kap.

Használati eset leírás részei	**Profiloldal készítése**
Követelmény	2. követelmény
A használati eset célja	Egy felhasználó profiloldal készítési igénnyel áll elő az adminisztrátor felé.

Előfeltételek	A rendszerben csak korábban igényt leadott felhasználókat rendelkezhetnek profiloldallal, tehát szükséges az elkészítéshez egy igény megléte	
Sikeres lefutás feltétele	Profiloldal készül a felhasználó részére	
Sikertelen lefutás feltétele	A profiloldal elkészítése visszautasításra kerül	
Elsődleges aktor	Adminisztrátor	
Másodlagos aktor	Igények adatbázisa	
Kiváltó esemény	Az adminisztrátor utasítást ad a hozzáférés elkészítésére	
Tartalmazott eset	Igény ellenőrzése	
Fő lépések	Lépések	Tevékenységek
	1	Az adminisztrátor utasítást ad a profiloldal elkészítésére
	2	A rendszer bekéri a felhasználó adatait
	3	Az adminisztrátor megadja a felhasználó adatait.
	4 **include:: Igény ellenőrzése**	A felhasználói igény ellenőrzésre került.
	5	A profiloldal elkészül
	6	A profiloldal elkészüléséről a felhasználó emailben értesítést kap.

Ezután elkészíthető a felhasznált lépések alkalmazásával az új, Igény ellenőrzése használati eset táblázat.

Használati eset leírás részei	**Igény ellenőrzése**	
Követelmény	1. követelmény, 2. követelmény	
A használati eset célja	A felhasználó által leadott igény ellenőrzése és pozitív visszajelzés.	
Előfeltételek	A rendszerben csak korábban igényt leadott felhasználókat kezelnek.	
Sikeres lefutás feltétele	Az igény visszaigazolása	
Sikertelen lefutás feltétele	Az igény meglétének hiánya miatti visszautasítás	
Elsődleges aktor	Igények adatbázisa	
Másodlagos aktor	-	
Kiváltó esemény	A rendszer ellenőrzést kér az Igények adatbázisától.	
Fő lépések	Lépések	Tevékenységek
	1	A rendszer ellenőrzést kér az Igények adatbázisától
	2	Az adattár ellenőrzi az igényeket.
	3	Az igény megléte visszaigazolásra kerül.

Kivételek	Lépések	Elágazó tevékenységek
	2.1	Az igények adatbázisa nem tartalmaz leadott igényt
	2.2	Az igény hiányának visszajelzése

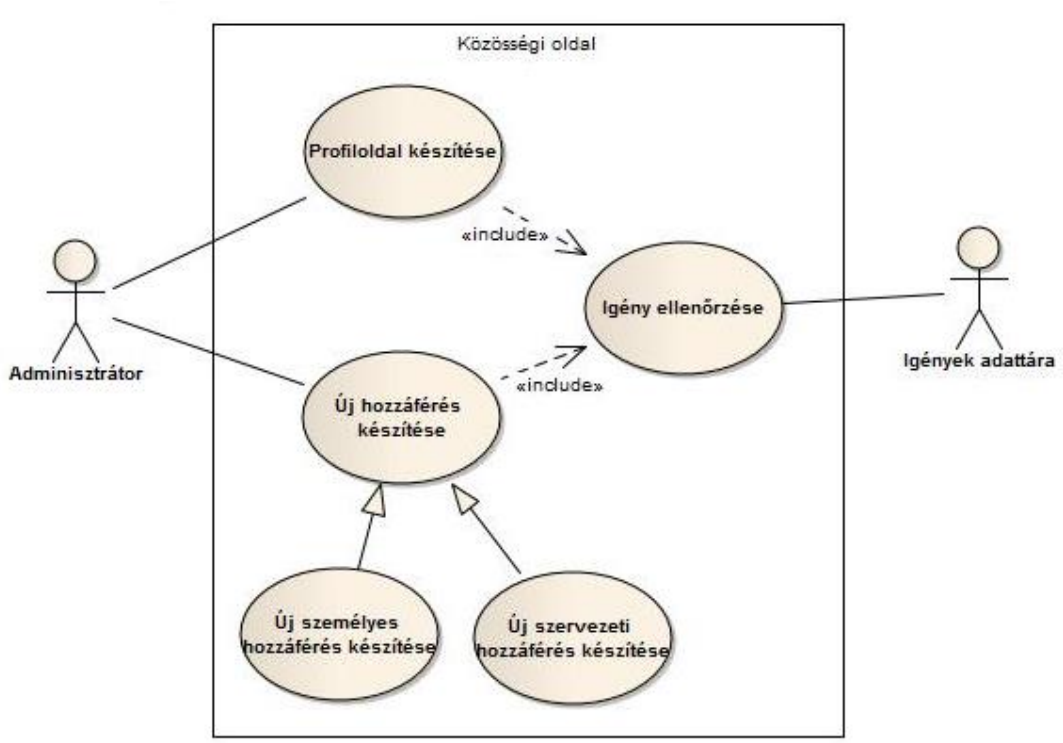
Az azonos lépések azonosítása a használati esetekben lehetővé teszi a másolások elkerülését. Ha valamiért a használati esetek ezen lépései változtatásra kerülnek, elég csak egy helyen változattani a leírást. A későbbiekben az implementáció során az adott használati eset lépések egyszer kerülnek implementálásra. Ezzel nem csak a tervek lesznek átláthatóbbak, de a fejlesztés ideje is csökkenthető.

Speciális esetek

A használati esetek tervezésekor előfordulhat, hogy a lépések meghatározása közben észrevesszük, egy eset csak kis mértékben tér el egy másik használati esettől. Az <<include>> kapcsolttal ellentétben, ahol a lépések teljes egészében a használati eset részeként jelennek meg, a lépések között csak kis változások, vagy kis tulajdonságbeli különbségek fedezhetőek fel. Objektum-orientált nyelven ilyen esetben több speciális esetről vagy egy ős esetről beszélünk.

Példáként vegyük a Közösségi oldalt megvalósító rendszerünket. Korábban tárgyaltuk az Új hozzáférés készítése használati esetet, amelynek célja egy felhasználó részére hozzáférés készítése egy korábban leadott igény alapján. Ha azonban a közösségi oldal lehetőséget ad különböző típusú hozzáférések elkészítésére, akkor lehetséges, hogy ezen hozzáférések elkészítését leíró használati esetek csak ki mértékben térnek el egymástól, például más típusú adat megadására kérik fel a felhasználót. Ilyenkor az egyes esetek csak speciális lépésekkel vagy tulajdonságokban különböznek az eredeti használati esettől. .Ilyenkor meg kell határozni az általános esetet, amely az új hozzáférés elkészítését célozza és speciális esetet, amelyek ebből származnak. Ilyenek lehetnek például a magánszemélyként létrehozott hozzáférés és a szervezet vagy cég számára létrehozott hozzáférés. A szervezetek számára létrehozott hozzáférés annyiban különbözik a magánszemélyek számára létrehozott hozzáféréstől, hogy képes lesz bizonyos költségek fejében hirdetéseket vagy rendezvényeket létrehozni a rendszerben.

Ilyenkor jelenik meg a specializációs kapcsolat a használati esetek között. A specializációs kapcsolat esetben van egy ős vagy általános használati eset, amelyből a speciális osztályok származnak. Az ős vagy általános használati esethez több speciális eset kapcsolódhat. A zárt hegyű nyíl mutatja a speciális esettől az általánosítást az ős eset felé. Az alábbi ábra mutatja a Közösségi oldalt megvalósító rendszerben a speciális eset jelölését.



12. ábra: Szervezeti típusú hozzáférést leíró használati eset jelölése a használati eset diagramon.

Az Új szervezeti hozzáférés készítése használati eset leírásában jól látszik, hogy mely meghatározásoknál kerültek speciális megfogalmazások a használati eset leírásába. Csak a speciális megfogalmazások hozzáadása szükséges az eredeti használati eset leíráshoz.

Használati eset leírás részei	Új szervezeti hozzáférés készítése	
Követelmény	1. követelmény	
A használati eset célja	Egy felhasználó **szervezeti** hozzáférési igénnyel fordul az Adminisztrátorhoz	
Előfeltételek	A rendszerbe csak korábban igényt leadott felhasználókat lehet regisztrálni, tehát szükséges a felhasználó hozzáférésének elkészítéséhez egy igény megléte	
Sikeres lefutás feltétele	Új **szervezeti** hozzáférés készül a felhasználó részére	
Sikertelen lefutás feltétele	Az új **szervezeti** hozzáférés elkészítése visszautasításra kerül	
Elsődleges aktor	Adminisztrátor	
Másodlagos aktor	-	
Kiváltó esemény	Az adminisztrátor utasítást ad a hozzáférés elkészítésére	
Általános használati eset	Új hozzáférés készítése	
Fő lépések	Lépések	Tevékenységek

	1	Az adminisztrátor utasítást ad a hozzáférés elkészítésére
	2	**A rendszer lehetőséget ajánl a hozzáférés típusának kiválasztására.**
	3	Az adminisztrátor kiválasztja a **szervezeti** hozzáférés típusát.
	4	A rendszer bekéri a felhasználó adatait.
	5	Az adminisztrátor megadja a felhasználó adatait.
	6 include:: Igény ellenőrzése	Az igények adatbázisa visszaigazolja a leadott igényt.
	7	Az új **szervezeti** hozzáférés elkészül
	8	Az új **szervezeti** hozzáférés elkészüléséről a felhasználó emailben értesítést kap.
Kivételek	Lépések	Elágazó tevékenységek
	6.1	Az igények adatbázisa nem igazolja vissza a **szervezeti hozzáférés készítésére** leadott igényt
	6.2	A **szervezeti** hozzáférés elkészítése visszautasításra kerül

Használati eset leírás részei	**Új személyes hozzáférés készítése**	
Követelmény	1. követelmény	
A használati eset célja	Egy felhasználó **személyes** hozzáférési igénnyel fordul az Adminisztrátorhoz	
Előfeltételek	A rendszerbe csak korábban igényt leadott felhasználókat lehet regisztrálni, tehát szükséges a felhasználó hozzáférésének elkészítéséhez egy igény megléte	
Sikeres lefutás feltétele	Új **személyes** hozzáférés készül a felhasználó részére	
Sikertelen lefutás feltétele	Az új **személyes** hozzáférés elkészítése visszautasításra kerül	
Elsődleges aktor	Adminisztrátor	
Másodlagos aktor	-	
Kiváltó esemény	Az adminisztrátor utasítást ad a hozzáférés elkészítésére	
Általános használati eset	Új hozzáférés készítése	
Fő lépések	Lépések	Tevékenységek
	1	Az adminisztrátor utasítást ad a hozzáférés elkészítésére

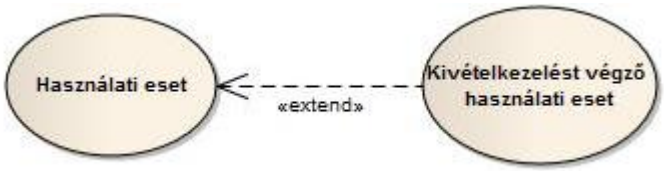
	2	**A rendszer lehetőséget ajánl a hozzáférés típusának kiválasztására.**
	3	Az adminisztrátor kiválasztja a **személyes** hozzáférés típusát.
	4	A rendszer bekéri a felhasználó adatait.
	5	Az adminisztrátor megadja a felhasználó adatait.
	6 include:: Igény ellenőrzése	Az igények adatbázisa visszaigazolja a leadott igényt.
	7	Az új **személyes** hozzáférés elkészül
	8	Az új **személyes** hozzáférés elkészüléséről a felhasználó emailben értesítést kap.
Kivételek	Lépések	Elágazó tevékenységek
	6.1	Az igények adatbázisa nem igazolja vissza a **személyes hozzáférés készítésére** leadott igényt
	6.2	A **személyes** hozzáférés elkészítése visszautasításra kerül

A használati eset általánosítás egy jól használható módszer a használati esetek létrehozására akkor, ha csak néhány speciális lépés kijelölésére van szükség. Azonban a specializációnál is figyelembe kell venni azt a tényt, hogy minden lépés, amelyet az általános eset tartalmaz, megjelenik a specializált esetekben is. Ugyanígy ha az általános használati eset kapcsolatban van egy külső aktorral vagy <<include>> kapcsolatban van jelen egy másik használati esettel, annak a kapcsolatnak érthetőnek kell lennie. Így például az Új hozzáférés létrehozása használati eset <<include>> kapcsolata az Igény ellenőrzése használati esettel kapcsolatot jelent az Új szervezeti hozzáférés készítése használati esettel is.

Ha mindezek nem jellemzőek az előzőleg speciális esetnek kikiáltott estekre, akkor azok nem specializált használati esetek. Lehetséges, hogy ilyen esetben a korábban tárgyalt <<include>> vagy a következő <<extend>> kapcsolatról van szó.

Az <<extend>> kapcsolat

Az UML a következőkben határozza meg két használati eset közötti <<extend>> kapcsolatot.



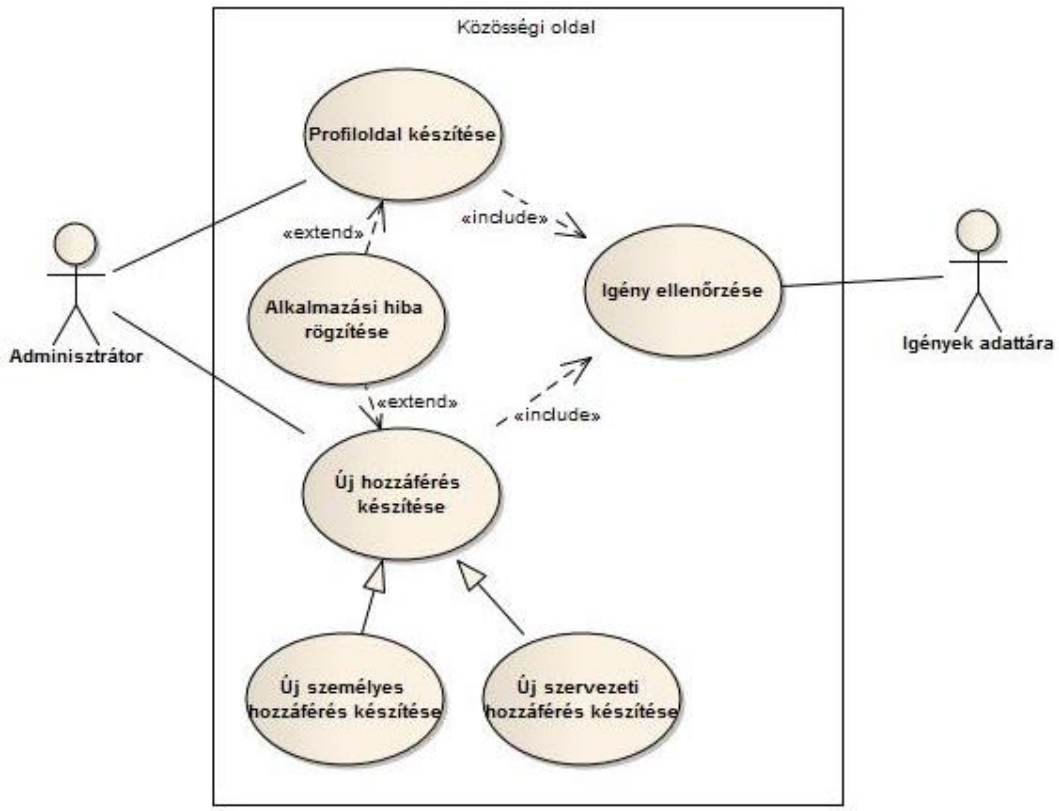
13. ábra: Két használati eset közötti <<extend>> kapcsolat jelzése.

Két használati eset közötti <<extend>> kapcsolat az <<include>> kapcsolathoz hasonlóan a használati eset teljes felhasználását jelenti. Azonban az <<include>> kapcsolattal ellentétben a kiterjesztés felhasználása opcionális, és egy feltétel meglététől függ. A Közösségi oldalt létrehozó rendszer esetében a lehetőség van a hozzáférési kérések visszautasításának rögzítésére. Ezeket a bejegyzéseket a felhasználókról gyűjtött jelentésekben tárolja a rendszer. Ezen rögzítések kezelésére extra lépés vezethető be az Új hozzáférés készítése használati esethez.

Használati eset leírás részei	**Új hozzáférés készítése**	
Követelmény	1. követelmény	
A használati eset célja	Egy felhasználó hozzáférési igénnyel fordul az Adminisztrátorhoz	
Előfeltételek	A rendszerbe csak korábban igényt leadott felhasználókat lehet regisztrálni, tehát szükséges a felhasználó hozzáférésének elkészítéséhez egy igény megléte	
Sikeres lefutás feltétele	Új hozzáférés készül a felhasználó részére	
Sikertelen lefutás feltétele	Az új hozzáférés elkészítése visszautasításra kerül	
Elsődleges aktor	Adminisztrátor	
Másodlagos aktor	-	
Kiváltó esemény	Az adminisztrátor utasítást ad a hozzáférés elkészítésére	
Tartalmazott eset	Igény ellenőrzése	
Fő lépések	Lépések	Tevékenységek
	1	Az adminisztrátor utasítást ad a hozzáférés elkészítésére
	2	A rendszer bekéri a felhasználó adatait
	3	Az adminisztrátor megadja a felhasználó adatait.
	4 **include:: Igény ellenőrzése**	A felhasználói igény ellenőrzésre került.
	5	Az új hozzáférés elkészül
	6	Az új hozzáférés elkészüléséről a felhasználó emailben értesítést kap.
Kivételek	Lépések	Elágazó tevékenységek

	4.1	Az igények adatbázisa nem igazolja vissza a hozzáférés készítésére leadott igényt
	4.2	A hozzáférés elkészítése visszautasításra kerül
	4.3	**A hozzáférés készítésének visszautasítása bejegyzésre kerül a felhasználóról készült jelentések közé**

A visszautasítás rögzítésének lépése ugyanúgy használható a profiloldal készítése használati eset kivételei között is. Ez az eset csak megfelelő feltételek között futhat le mindkét használati esetnél. Csak akkor, ha az új hozzáférés készítése vagy a profiloldal készítése visszautasításra kerül.



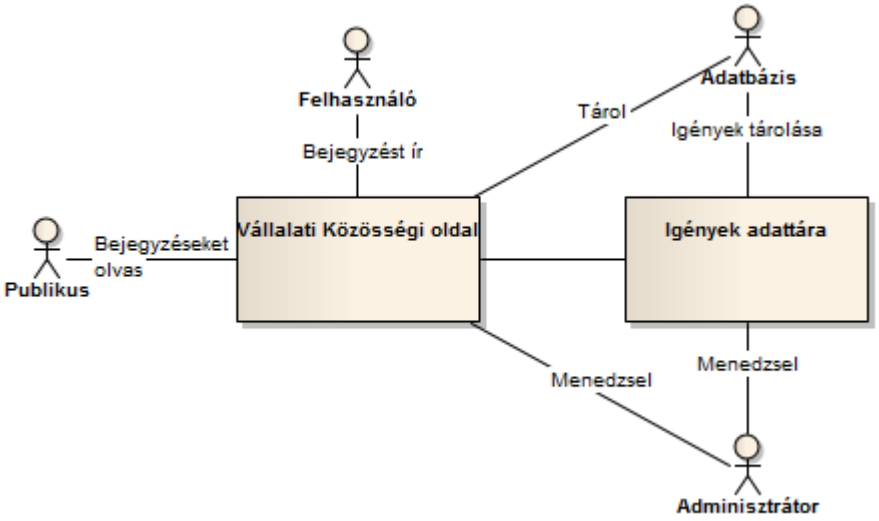
14. ábra: Az <<extend>> kapcsolatok szerepe a rendszer használati esetei között.

Új opcionális használati esetként jelenik meg az Alkalmazási hiba rögzítése. Ahogyan a nevéből adódik az alkalmazás működése során keletkező hibákat rögzíti. A rendszerben az Új hozzáférés készítése és a Profiloldal készítése használati esethez kapcsolódik. A kapcsolat azonban opcionális lefutású

<<extend>> kapcsolat, és csak akkor futnak le a használati eset lépései, ha az alkalmazás futása közben hiba történik.

Használati eset áttekintő diagram

Nagyobb méretű rendszereknél, ahol a használati esetek száma már megnehezíti a rendszer egyszerű megértését a használati eset diagram alapján, célszerű egy úgynevezett használati eset áttekintő diagram készítése. Az áttekintő diagram lehetőséget nyújt a rendszer felsőbb szintű bemutatására.



15. ábra: A vállalati Közösségi oldalt megvalósító rendszer áttekintő diagramja.

A használati eset áttekintő diagram elnevezése némileg félrevezető, hiszen általánosságban nem tartalmaz használati eseteket. A diagram leginkább a rendszer környezetéről, kapcsolatairól ad felvilágosítást.

3. Használati eset diagram és a mintarendszer leírása

A modellezés első lépésében a feladat a szakterület és a létrehozandó szoftver megismerése, a funkcionalitások azonosítása. Ennek során a szakterület képviselőivel úgynevezett üzleti interjúkat kell készíteni. A felmérés után az interjú lejegyzésével egy szöveges leírást kapunk, amelyet táblázatos formában, strukturált szövegként kell feldolgozni.

Ez alapján történik meg a rendszer funkcionális követelményeinek meghatározása, azonosítása, amelynek első lépésében az üzleti folyamatok, illetve használati esetek meghatározására a feladat.

A rendszerek egy cél szolgálatában jönnek létre. Ezek a célok funkcionális és nemfunkcionális követelményekként azonosíthatók. A funkcionális követelmény nem más, mint a tulajdonképpeni feladat, amit a rendszernek el kell végeznie. A nemfunkcionális követelmény az egyéb megszorításokat takarja, mint például mennyi ideig futhat egy kérés a rendszerben, mennyi idő áll rendelkezésre a válaszadásra.

A funkcionális követelmények esetében beszélhetünk folyamatokról és használati esetekről. A folyamatok lehetnek üzleti folyamatok, ha információs rendszerekkel kapcsolatban merülnek fel, illetve rendszerfolyamatok, ha technikai területeken létező folyamatokról beszélünk. A folyamatokat néha alfolyamatokra bontjuk tovább. A folyamatok egyes munkalépései használati esetek segítségével írhatók le. Az üzleti folyamtok és használati esetek nagyon hasonlóak egymáshoz, nagyrész ugyanazokkal az eszközökkel is írják le őket, leginkább az ellipszisként megjelenő UseCase konstrukcióval. Azonban lényeges különbségek is vannak köztük.

A TicketFirst rendszerre vonatkozó esettanulmányunk Koncertlátogatás üzleti folyamatához tartozó interjúja a következőkben olvasható:

A koncertlátogatás három szakaszból áll: jegyvásárlásból, a koncerthallgatásból, és a bónuszpontok utólagos jóváírásából. A jegyvásárlás során a felhasználó az interneten bejelentkezik a rendszerbe, azonosítja magát a nevével és a jelszavával. Kiválasztja azt a koncerttípust ami iránt érdeklődik, majd megadja, hogy melyik időpontban, és helyszínen rendezett koncertek érdeklik. A rendszer megjeleníti a kérésének megfelelő koncerteket, amelyek közül vásárló választ. Ekkor megjelennek a lehetséges jegytípusok (álló, ülő, sor, szék, jegyár). A vásárló ezek közül választ, majd további adatok megadása történhet. A vásárlás megerősítése után a jegyek kifizetése történik, amelyre többféle lehetőség van (hitelkártya, bankszámla). A fizetéshez további adatokat kell megadnia, mint például a hitelkártya

száma. A rendszer ezután rögzíti a vásárlást, kinyomtatja a jegyet, ha szükséges, illetve számlát készít, ha kérték.

A koncert lebonyolítása a jegy ellenőrzésével kezdődik. Az ellenőrzés történhet automatikusan vagy a személyzet segítségével. Itt a jegy száma alapján azonosítják, hogy a jegy megfelelő e az adott rendezvényre. A következőkben megtörténik a néző azonosítása a rendszerben a jegy vagy a bónuszkártya alapján. Megfelelő adatok esetén megtörténik a jegykezelés és a beléptetés során a fogókapu nyitásával a nézőt beengedik a helyszínre

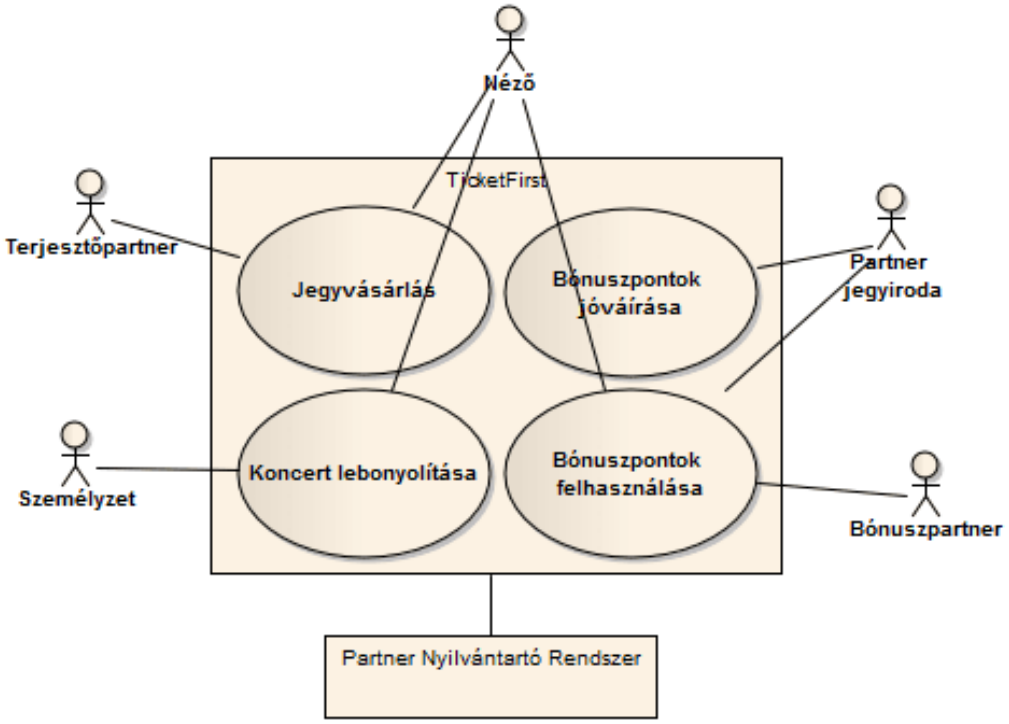
A koncert fajtájától, a jegy típusától illetve árától függően a jegyvásárláshoz és a koncerten való részvételhez, és a koncerthelyszínen történő vásárláshoz bónuszpontok kapcsolódnak. Ezek jóváírása a koncert lebonyolítása után történik meg. A pontok mennyiségének függvényében a néző státusza is változhat.

Az itt látható táblázatos forma egy lehetőség az üzleti folyamatok szabványos leírására. Az irodalomban ez többféle módon megjelenhet, a főbb jellemzői viszont azonosak az egyes táblázattípusoknak

Azonosító ÜF - KL - TF	Név Koncertlátogatás
Rövid leírás Koncertlátogatás teljes folyamat a jegyvásárlástól a bónuszpontok felírásáig	
Érintett aktorok 1. Néző, 2. TicketFirst	
Kiváltó esemény A néző el szeretne menni egy koncertre	
Előfeltétel nincs	
Standard lefutás 1. Jegyvásárlás 2. Koncert lebonyolítása 3. Bónuszpontok jóváírása	Kivételek és alternatívák a, a néző a bónuszpontkártyáról fizeti a jegyet, a 3. pont kimarad b, a bónuszkártyán a néző státuszának léptetése szükséges,mert határt ért el
Utófeltétel Mindhárom folyamat utófeltételeinek együttese	
Eredmény A koncert lezajlik, az irodának bevétele származik belőle	
Gyakoriság napi 3000	
Utalások későbbiekben kerül kitöltésre	
Megjegyzések, nyitott kérdések nincs	

Üzleti folyamat leltár

Az üzleti folyamatok felsorolásával tehát meghatározhatók a rendszer határai. A folyamatleltárral vizuálisan is érzékeltethető a rendszer és környezetének elkülönítése. Az ábrán látható a TF rendszerhez tartozó folyamatleltár.



16. ábra: A TicketFirst rendszer folyamatainak leltára

A folyamatleltárban nem tüntetjük fel az aktorok közötti kapcsolatokat, itt a rendszer és a résztvevők kapcsolatára koncentrálunk. Az ábrán az ellipszisekben találhatóak a TicketFirst rendszer főbb üzleti folyamatai, hozzájuk összekötővel jelölve kapcsolódnak az aktorok. A rendszerhez kapcsolódik egy Partner Nyilvántartó (BestParts), mint aktor, amely a partner jegyirodák vagy szervezőirodák adatainak adminisztrálására szolgál.

Használati esetek, használati eset leltár

A használati esetek az üzleti folyamatok részeit képezik. A legtöbb esetben nem egyszerű eldönteni, hogy üzleti folyamatról vagy használati esetről van e szó, de ahhoz hogy a tervezési lépésekben továbbléphessünk, ezt mindenképpen meg kell tennünk. Nézzünk egy példát.

A Belépéshez a következő lépések tartoznak:

1. A Néző a jegyét ellenőrizteti saját maga az automata rendszerrel, vagy személyzet segítségét kéri, aki ezt megteszi helyette.

2. A Néző azonosítása a bónuszkártyája vagy a hitelkártyája alapján.
3. A forgókapu nyitása.

A lépések azonosítása után meg kell nézni, hogy az üzleti folyamat vagy a használati eset szabályai érvényesek-e a folyamatra. Mivel folyamat, ahogyan látszik több lépésből áll, de ez nem zárja ki azt hogy használati eset lehessen, hiszen egy használati eset is állhat részekből. Nem fut sokáig, pár perc alatt lezajlik a beléptetés. Általában egy résztvevő vesz részt a folyamatban, ez pedig a Néző, illetve a személyzet egy tagja esetleg, ha a Néző segítséget kér. A rendszerből a beléptetési alrendszer áll kapcsolatban az aktorral. A folyamat automatikusan mehet végbe, ha a Néző végrehajtja a kéréseket. A felsorolt nézőpontok alapján tehát a folyamat használati esetnek tekinthető.

A használati esetek is leírhatók táblázatos formában. A tárgyalt használati eset táblázata a következőképen néz ki:

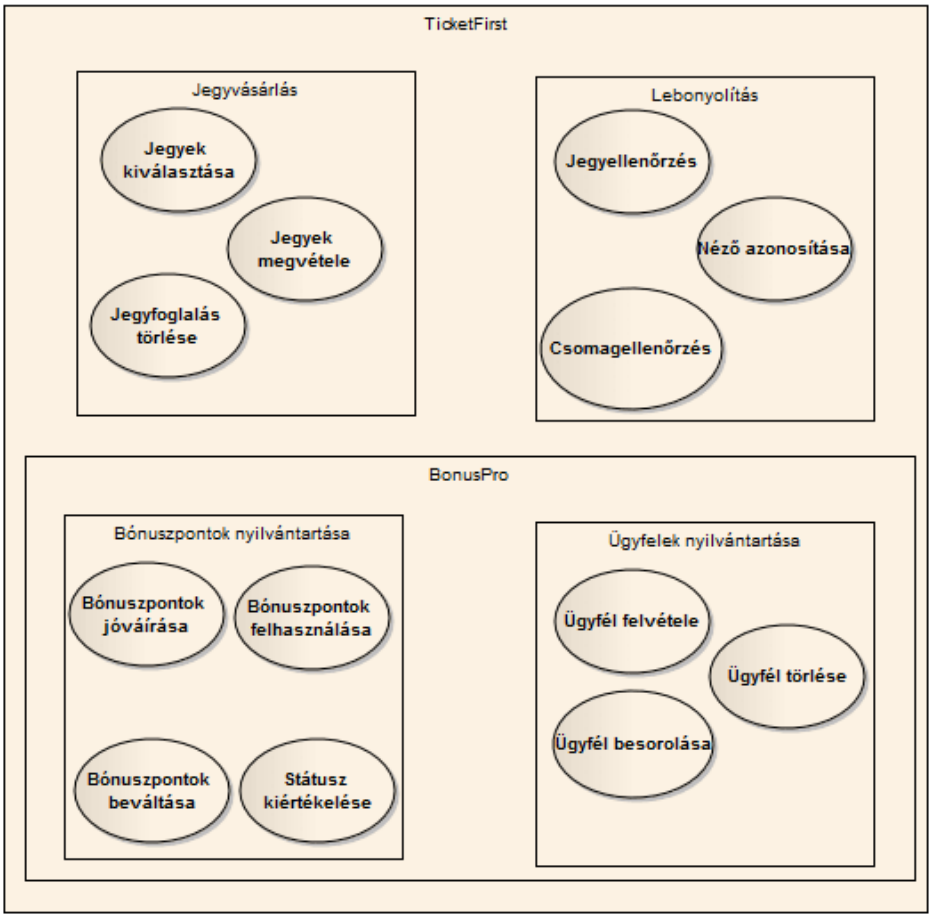
Azonosító HE - L - BLA	Név Belépés
Rövid leírás A koncertre érkező néző jegyének ellenőrzése, néző azonosítása	
Érintett aktorok 1. Néző, 2. TF.Jegyvásárlás	
Kiváltó esemény A néző be szeretne lépni a koncerthelyszínre, behelyezi a jegyet az automatába	
Paraméter Dátum, idő, helyszín	
Standard lefutás 1. Jegyek ellenőrzése 2. Jegykezelés 3. Belépés a forgókapun	Kivételek és alternatívák a, A néző több jeggyel rendelkezik az adott koncertre, ilyenkor több jegyet kell ellenőrizni b, A jegyek ellenőrzése nem sikerül (sérült jegy), a személyzet segítségét kell kérni
Utófeltétel Nincs	
Eredmény A jegyek ellenőrizve, érvényesítve	
Átlagos időtartam 5 perc	
Utalások használja: ÜF-TF-BP, illetve GUI-B-1..3	
Megjegyzések, nyitott kérdések nincs	

A használati eset táblázat nem sokban különbözik az üzleti folyamat táblázattól. Eltérés a következőkben van:

- Az eredmény itt nem az aktor számára kitűzött célt tartalmazza, hanem pl, egy kimenete, így az adott folyamatban a jegyek érvényesítésének sikeressége.
- A használati eseteknél nem a gyakoriság a lényeges, hanem a használati eset lefutásának hossza.
- A használati eset lefutásához bizonyos adatokra, paraméterekre lehet szükség. Ebben az esetben például koncert adataira, amelyre a Néző jegyet vásárolt.

Ahogyan az üzleti folyamat diagramnál, itt is elmondható, hogy nem minden esetben ugyanilyen formátumú használati eset diagramokkal találkozunk, de a fő címkék mindegyiknél megjelennek.

A rendszerben lévő használati esetek összességéről használati eset leltár hozható létre. Ebben a rendszerben megjelenő használati esetek, üzleti folyamatok az alrendszerek alá vannak besorolva.



17. ábra: A TicketFirst rendszer használati eseteinek leltára

Észrevehető, hogy a kialakítandó rendszer tervezése során ahogyan lépünk előre, annál többet tudunk meg a funkcionalitásokról. Az üzleti interjúban még nem szereplő részetekre a modellezés során derül fény, így a résztvevőkkel több alkalommal is szükség lehet egyeztetésre. Természetesen nem azonnal tudunk meg minden a számunkra lényeges információt, a szoftverfejlesztés folyamata nagyobb

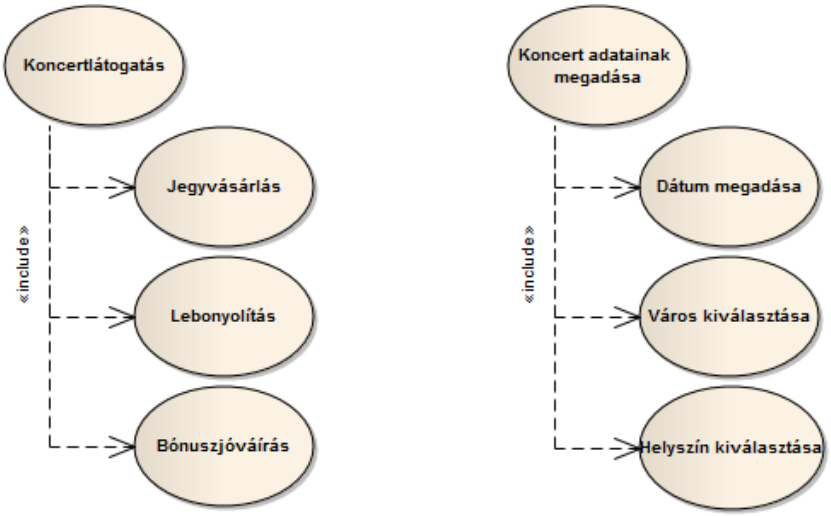
projektek esetében nem lineáris, gyakran vissza kell térnünk az előző szakaszokhoz. Így lehetséges, hogy a már előzőleg véglegesnek ítél használati eset leíráson (táblázaton) is változtatnunk kell.

Függőségek funkcionalitások között

A használati esetek és az üzleti folyamatok között is meghatározhatunk bizonyos függőségeket. Az UML-ben leginkább az „include” és az „extend” kapcsolatok jelennek meg. Ezek segítséget nyújtanak a modellezőnek abban, hogy átlássa az egyes folyamatok egymásra épülését, a köztük lévő kapcsolatrendszert. A későbbi modellek esetében is hasznos ennek tisztázása ebben a fázisban.

Magábanfoglalás – include

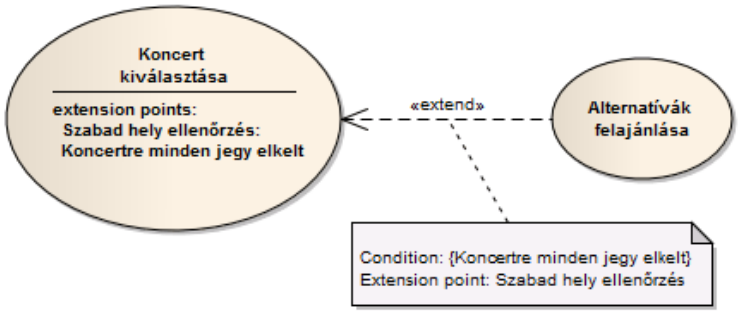
Az alábbi ábrán látható a Koncertlátogatás üzleti folyamat és a részét képező használati esetek magában foglalás relációja. A belefoglalt esetek magukban is értelmesek, a folyamat közben többször is lefuthatnak, de legalább egyszer szerepelniük kell. Így a dátum megadása többször is előfordulhat a koncert adatainak megadása közben, ha a Néző több időpontra kíván jegye foglalni, esetleg ő is szeretné megnézni a koncertet, de ajándékba is szán jegyet.



18. ábra: Include kapcsolat a folyamatok között

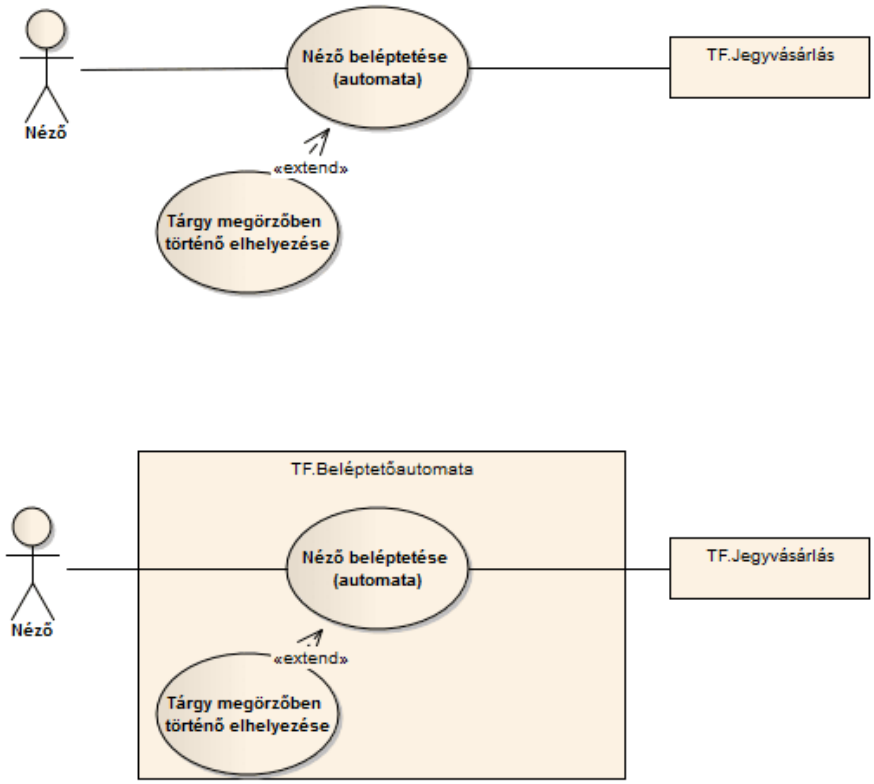
Kiterjesztés – extend

A kiterjesztési pontra olyan esetben van szükség, ha a folyamat egyes lépései között opcionális elemek vannak. Ilyen például, ha a koncert kiválasztása nem lehetséges, mert a koncertre minden jegy elkelt, a rendszer alternatív koncerteket ajánl fel. Az alternatívák felajánlása folyamat tehát nem minden esetben fut le, csak abban az esetben ha a koncert kiválasztása standard lefutása valamiért megakad.



19. ábra: Kiterjesztési pont a Koncert kiválasztása folyamatban

Egy nagyobb rendszer esetében a használati esetek dokumentációja egy idő után áttekinthetetlenné válhat. Több diagramfajta is alkalmas ennek könnyebbé tételére. Egy ilyen vizuális megjelenítés a következőképen nézhet ki:



20. ábra: Néző beléptetése folyamat és a kapcsolódó használati esetek és aktorok vizuális megjelenítése

Látható, hogy a Néző beléptetése folyamat a Tárgy megőrzőben történő elhelyezésébe használati esettel van kiterjesztve. Kapcsolódik hozzá még egy Néző aktor és a TF.Jegyvásárlás alrendszer. Az alsó ábrán feltüntetésre került az alrendszer is ami a folyamatot megvalósítja.

4. A rendszer dinamikájának modellezése: tevékenységdiagramok a használati esetek alapján

A használati esetekkel meghatározhatóak a rendszer funkciói. A funkciók működésének részleteit illetve ezek folyamatát a tevékenységdiagramokkal írhatjuk le. A tevékenységdiagramok műveletek egymásutániságával írják le a rendszerben lejátszódó folyamatokat. Az eddig a rendszerben azonosított használati esetek tevékenységdiagramokkal részletezhetőek. A tevékenységdiagramok jól használhatóak üzleti folyamatok leírására is. Az üzleti folyamatok és a használati esetek tevékenységfolyamok leírására használhatóak. A használat esetek a rendszerben lefutó tevékenységfolyamok leírására szolgálnak, míg az üzleti folyamatok a rendszer határain is túlnyúlhatnak. Az üzleti folyamatok rendszerek felett állnak, míg a használati esetek egy rendszeren belül határozhatóak meg. A használati esetek rövid idő alatt és különösebb működés közbeni interakció nélkül futnak le. Az üzleti folyamatok megszakíthatóak, gyakran napokig vagy hónapokig zajlanak, futás közben folyamatos párbeszédet tartanak fenn.

A folyamatok az UML-ben tehát tevékenységdiagramokkal modellezhetőek, amely egy grafikus modellezési forma. Mint ilyen egyszerű értelmezést tesz lehetővé, egyértelművé teszi kívülállók számára is a folyamatok lefutását. Az üzleti folyamatok lefutásának leírására többek között gyakran használják az üzleti életben az üzleti folyamat diagramokat. A tevékenységdiagramokon alapuló modellező eszköz és diagramtípus az üzleti folyamat diagram (BPD – Business Process Diagram), amely leginkább az üzleti területen tevékenykedő menedzsereket szolgálja.

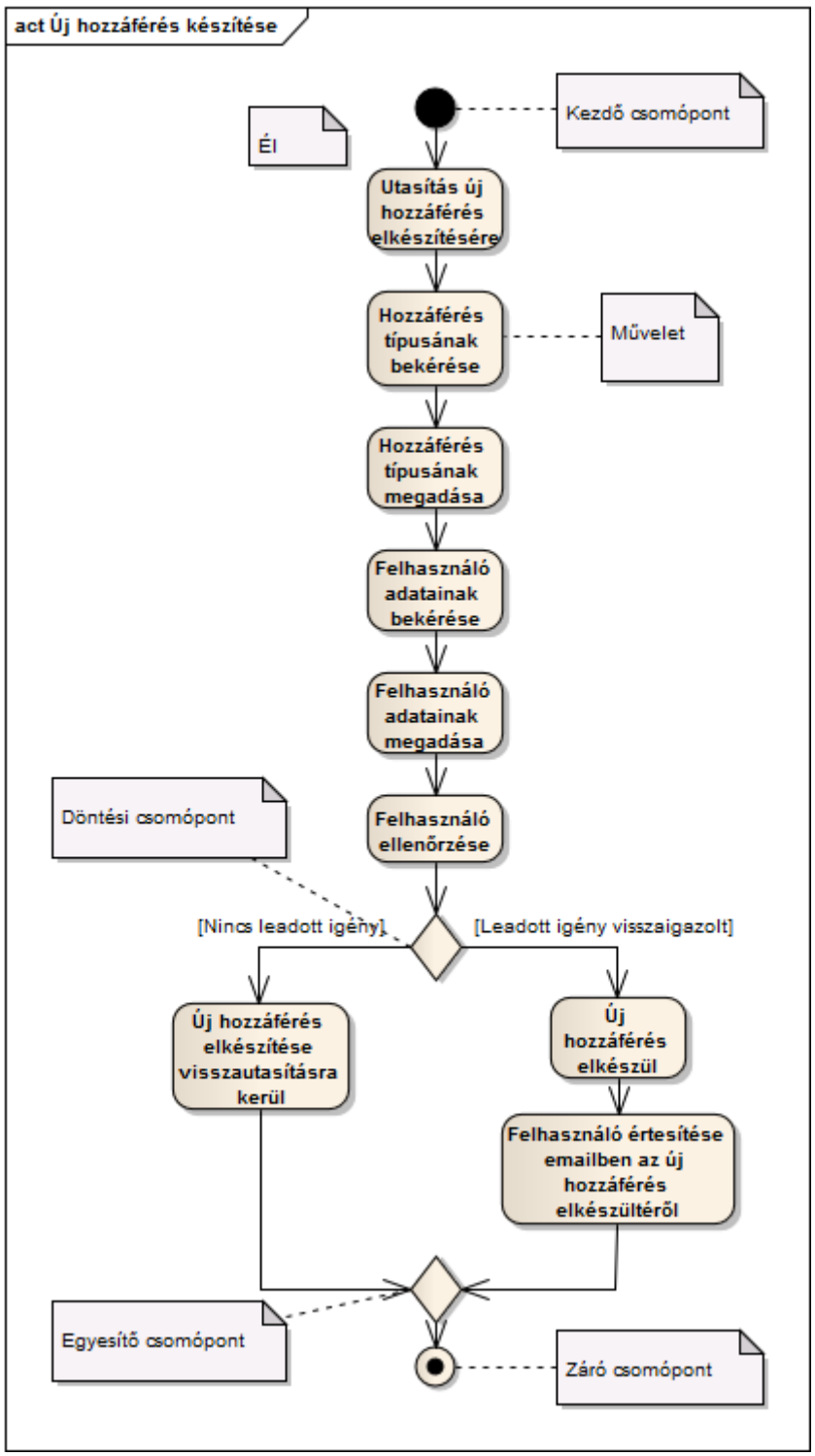
A tevékenységdiagramok a rendszer modelljében a folyamat nézetet képviselik. A jól ismert folyamatmodellezési jelölésrendszerben használt jelöléseket alkalmazzák, ezért jól használhatóak tágabb körben is. A tevékenységdiagramok eredete az UML állapotdiagramokhoz, az adatfolyam diagramokhoz és a Petri hálóhoz vezethetőek vissza.

Használati esetek és tevékenységdiagramok kapcsolata

A használati esetekben leírt lépések átvezethetőek tevékenységdiagramok műveleteire, tevékenységeire. Példaként vegyük az előző fejezetben tárgyalt Új személyes hozzáférés elkészítése használat eset átvezetését tevékenység diagramra. Az alábbi táblázatban látható az Új személyes hozzáférés készítése üzleti folyamat táblázata.

Használati eset leírás részei	Új személyes hozzáférés készítése	
Követelmény	1. követelmény	
A használati eset célja	Egy felhasználó **személyes** hozzáférési igénnyel fordul az Adminisztrátorhoz	
Előfeltételek	A rendszerbe csak korábban igényt leadott felhasználókat lehet regisztrálni, tehát szükséges a felhasználó hozzáférésének elkészítéséhez egy igény megléte	
Sikeres lefutás feltétele	Új **személyes** hozzáférés készül a felhasználó részére	
Sikertelen lefutás feltétele	Az új **személyes** hozzáférés elkészítése visszautasításra kerül	
Elsődleges aktor	Adminisztrátor	
Másodlagos aktor	-	
Kiváltó esemény	Az adminisztrátor utasítást ad a hozzáférés elkészítésére	
Általános használati eset	Új hozzáférés készítése	
Fő lépések	Lépések	Tevékenységek
	1	Az adminisztrátor utasítást ad a hozzáférés elkészítésére
	2	**A rendszer lehetőséget ajánl a hozzáférés típusának kiválasztására.**
	3	Az adminisztrátor kiválasztja a **személyes** hozzáférés típusát.
	4	A rendszer bekéri a felhasználó adatait.
	5	Az adminisztrátor megadja a felhasználó adatait.
	6	Az igények adatbázisa visszaigazolja a leadott igényt.
	7	Az új **személyes** hozzáférés elkészül
	8	Az új **személyes** hozzáférés elkészüléséről a felhasználó emailben értesítést kap.
Kivételek	Lépések	Elágazó tevékenységek
	6.1	Az igények adatbázisa nem igazolja vissza a **személyes hozzáférés készítésére** leadott igényt
	6.2	A **személyes** hozzáférés elkészítése visszautasításra kerül

A következő ábrán látható a használati eset tevékenységdiagram jelölése. A tevékenységdiagram áttekinthetőbbé teszi az egyes lépések közötti kapcsolatokat. Jól látható az elágazás, ahol az igénylő ellenőrzése megtörténik.



21. ábra: A tevékenységdiagramokkal a rendszer dinamikáját modellezzük. A tevékenységdiagram alapelemei láthatóak a modellezett Új személyes hozzáférés készítése folyamatban.

A tevékenységfolyam a kezdő csomóponttal kezdődik, amelyet teli körrel jelölnek. A tevékenységdiagram végét a záró csomópont jelöli, amely két koncentrikus körrel egy belső teli körrel jelölnek. A kezdő és a záró csomópont között helyezkednek el műveletek, amelyeket lekerekített téglalapok jelölnek. A műveletek a tevékenységdiagramok főszereplői, amelyek az egyes lépéseket írják le, így például a Hozzáférés típusának kiválasztása, a Felhasználó adatainak megadása, stb. Egy művelet lehet egy viselkedés leírása, egy számítás, vagy a folyamat bármely fontos lépése.

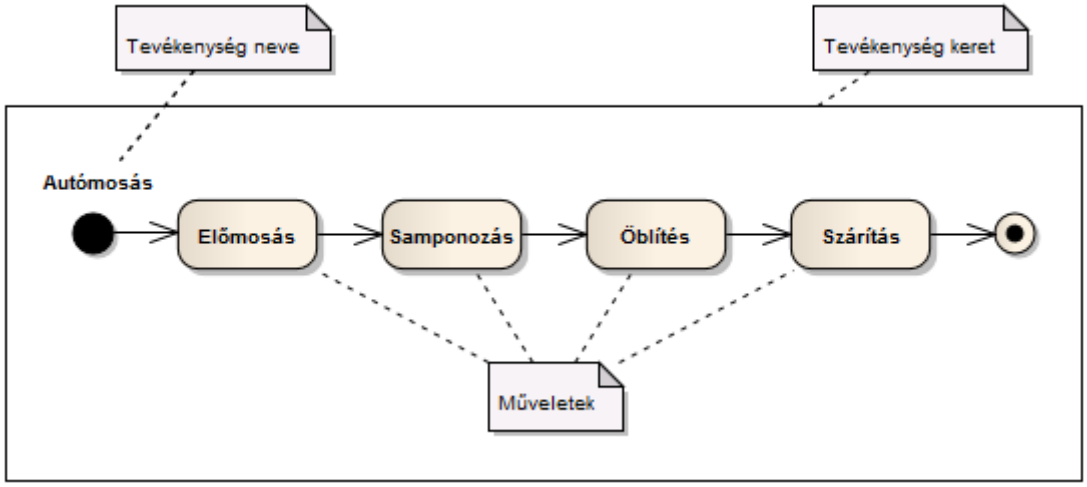
A tevékenységfolyamot a diagramon a nyilak vagy utak jelölik. A nyílvégek jelölik a folyam irányát egyik művelettől a másikig. A műveletbe érkező nyílvéget bemenő élnek (incoming edge), a kimenő vonalat kimenő élnek (outgoing edge) nevezik. A nyilak kötik össze a műveleteket, így alkotva meg a teljes tevékenységfolyamot.

A rombusz jelöli a döntési csomópontokat (decision), amelyek az if-else feltételes operátornak felel meg a programozásban. Ebben az esetben a döntési csomópontnak két kimenő éle van, amelyen feltételek jelennek meg. Ezeket a feltételeket úgy kell megadni, hogy a döntés alapján egyértelmű legyen a folyamat iránya. A második rombusz alakú csomópont az egyesítő csomópont, amely a döntési csomópontból kimenő éleket egyesíti, és a feltétel végét jelöli.

A tevékenységdiagramban a folyam folyását jelenti a műveletek egymásutánisága: egyik művelet befejeződését követően a folyam átadja a vezérlést a következő műveletnek.

Műveletek a tevékenységfolyamban

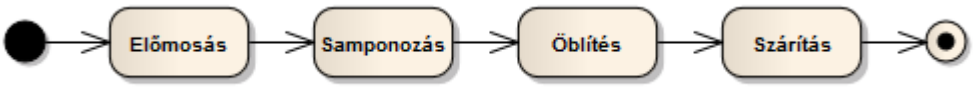
A műveletek a tevékenységdiagramok aktív lépései. A műveletek lehetnek számítások, vagy feladatok, mint például egy adat beírása egy táblázatba. A művelet a tevékenység, mint folyamat egy részegysége jelenik meg. Így például egy autó mosása során a műveletek az előmosás, samponozás, öblítés és szárítás. Ennek az egyszerű autómosási feladatnak a lépései láthatóak az alábbi ábrán.



22. ábra: Műveletek lépései a tevékenységben

A tevékenység a diagramon keretbe van foglalva. A keretet egy lekerekített téglalap adja, amelyet tevékenység keretnek hívnak. A tevékenység keretbe foglalásának az előnye, hogy ha a későbbiekben hivatkozunk a tevékenységre, akkor azt a tevékenységnek a keret bal felső részébe írt egyéni nevével tudjuk megtenni egyértelműen.

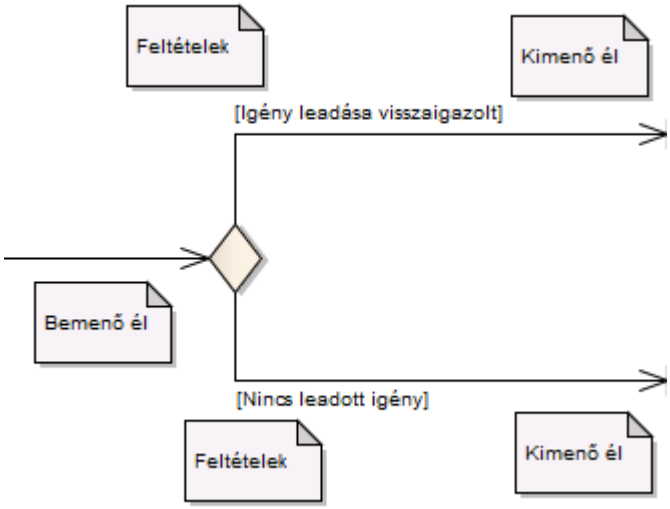
A tevékenység keretet nem kötelező megadni, enélkül is modellezhetjük a tevékenységfolyamot.



23. ábra: Az autómosás tevékenységfolyama tevékenységkeret nélkül ábrázolva

Döntési csomópontok és egyesítő csomópontok a tevékenységfolyamban

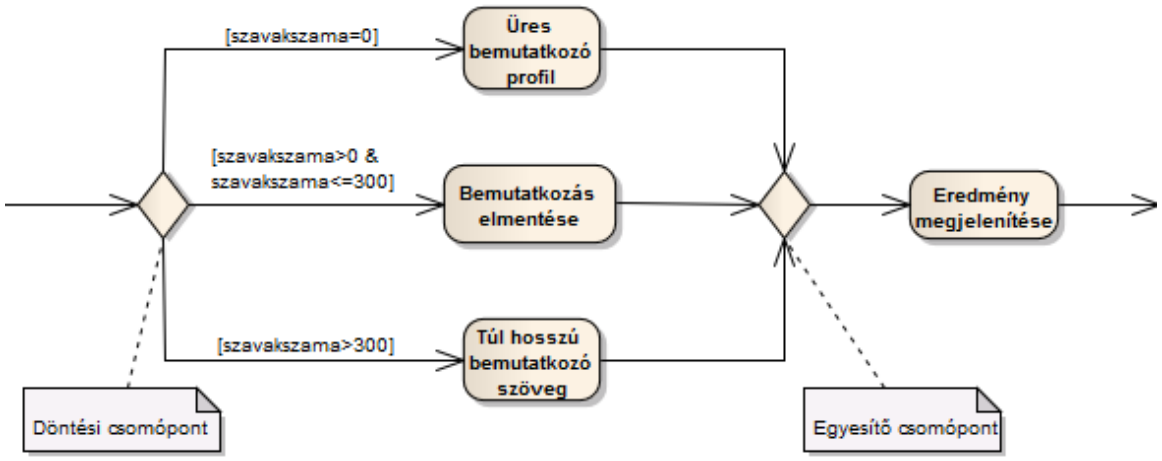
Ha egy tevékenységfolyam több felé ágazhat egy feltételtől függően, ott döntési csomópontokat kell alkalmazni. A döntési csomópontok csak a feltételek megadásával értelmezhetőek, hiszen ezek a feltételek adják meg, hogy a folyam melyik műveletek felé halad tovább. A döntési csomópontokat rombuszok jelölik. Egy döntési csomópontnak egy bemenő és több kimenő éle van. A kimenő éleken jelennek meg a döntési feltételek, szögletes zárójelben elhelyezve. A fennálló feltételeknek megfelelően folyatatódnak az egyes műveletek a tevékenységfolyamban.



24. ábra: A döntési csomópont jelölése a tevékenységfolyamban. A folyam csak egy úton folyik tovább a csomópont után.

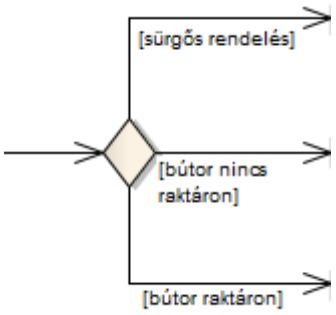
Vannak egyszerű döntési feltételek, ahol csak az a döntés tárgya, hogy a feltétel igaz vagy hamis. Ilyen, például ha a rendszerben azt vizsgáljuk, hogy egy szereplő rendelkezik egy rendszermodulhoz való hozzáférési jogosultsággal vagy sem. Egy ilyen feltétel esetén a [jogosultsággal rendelkezik] ág aktív. Olyan feltételt is lehetséges megadni, ahol egy összeg vagy szám egyenlősége illetve azt adott helyzethez való viszonya alapján történik meg a döntés. Ha pl. egy személyes bemutatkozás írása esetén a szavak száma nagyobb, mint 300, akkor a [szavak száma> 300] feltétel teljesül.

A szétválasztott folyamok egyesítő csomópontban találkoznak újra. Ez az egyesítő csomópont jelenti a feltételes eset végét. Az egyesítő csomópontokat szintén rombusszal jelölik, de a döntési csomóponttal ellentétesen több bemenő és egy kimenő ággal rendelkeznek.



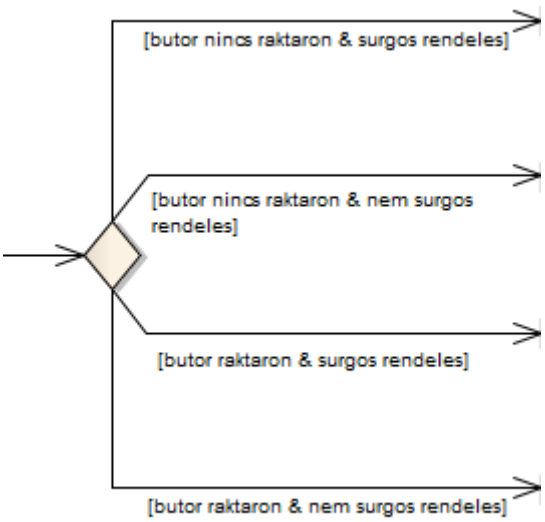
25. ábra: Szavak számától függő válasz a rendszerben. A folyam mindig a megfelelő feltétel irányába folyik tovább.

A döntési csomópontok megalkotásánál figyelni kell arra, hogy a feltételek egyértelműen megadják, mely ágon megy tovább a tevékenység. Nem engedhetőek meg egymást átfedő folyamágak. Vegyünk példának egy bútorrendelést: lehetséges, hogy az éppen megrendelt bútor raktáron van, de az is hogy éppen kifogyott. Emellett a megrendelő jelölheti sürgősnek a megrendelést. Ezt a tevékenységrészt ábrázolhatjuk a következőképen:



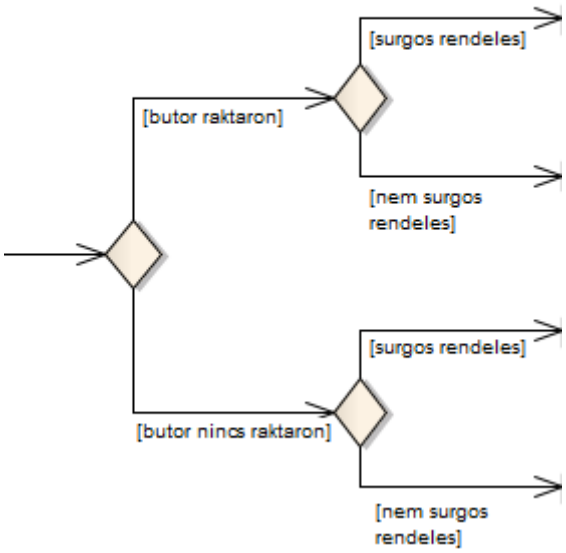
26. ábra: Több ág is aktívvá válhat. A tevékenység modellje javításra szorul.

Ilyenkor sérül az egyértelmű útválasztás feltétele, hiszen ha egy rendelés van raktáron és sürgős, akkor nem lehet eldönteni, hogy a folyam merre halad. A modell a következőképpen javítható:



27. ábra: A tevékenység javított modellje

Többféle modell megalkotása is lehetséges. Ilyen esetben mindegyik használható, ha megfelel a korábban felsorolt feltételeknek. A modellezőre, tervezőre van bízva, hogy melyiket használja.



28. ábra: Javított modell egy másik változata

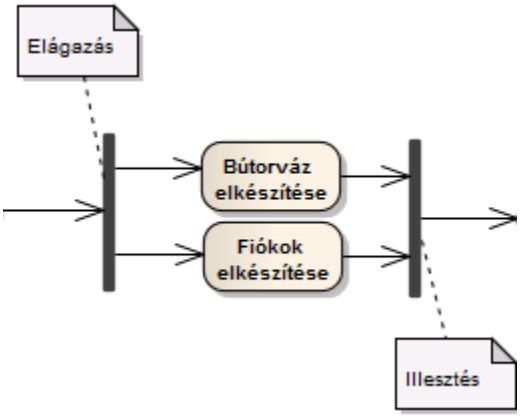
A rendszerek modellezésénél sok múlik a tervező személyén, előzetes tapasztalatain, a megrendelő kérésein. Nem meglepő, hogy ha két különböző ember vagy csapat készít modellt ugyanarról a rendszerről, az nagyon kismértékben várható hogy azonos lesz. Az azonosság hiánya azonban nem jelenti azt, hogy bármelyik is rossz, egyszerűen a különböző logika, tapasztalat és fennálló feltételek más eredményt adnak. Érdemes tehát kellő rugalmassággal hozzáállni egy rendszer tervezéséhez, főleg ha több embert is érintő projektmunkáról van szó.

Párhuzamosan futó műveletek a tevékenységfolyamban

A rendszerek futása közben, vagy bármely szervezeti tevékenység esetén lehetnek olyan műveletek, amelyek egymástól annyiban függetlenek, hogy egymással párhuzamosan is futhatnak. Így például két asszisztens munkáját végezve nem függ egymástól, ha az egyik iratokat fénymásol, a másik szerződéseket készít. A párhuzamos futó műveletekre veszünk példát az alábbiakban. Tekintsük a következő műveleteket:

- igények felmérése
- igények egyeztetése
- tervek elkészítése
- bútorváz elkészítése
- fiókok elkészítése
- helyszíni összeszerelés

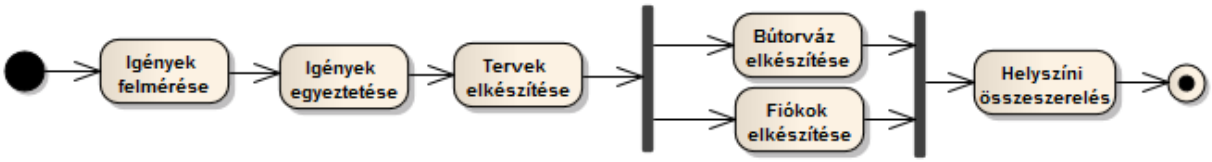
A műveletek listájából felfedezhető, hogy az egyes lépéseknek nem kell feltétlenül egymás után, szekvenciálisan lefutniuk. Azonosíthatóak párhuzamosan végezhető műveletek is. Ezeket a párhuzamosan futó műveleteket elágazással (fork) és illesztő (join) csomópontokkal kezeljük. A párhuzamosítás a jelölésben is jól értelmezhető: két párhuzamos vonal jelöli.



29. ábra: Bútorkészítési tevékenység párhuzamosan futó műveletei

Az elágazás után következő műveletek egymás mellett futnak. A két ágon a folyamok egyszerre indulnak el. Így a bútor vázának elkészítése és a fiókok elkészítése párhuzamosan történik. A döntési csomóponthoz hasonlóan a kimenő élek száma nem csak 2, hanem több is lehet. Az illesztő csomópont feladata, hogy szinkronizálja a beérkező folyamokat: csak akkor lép ki belőle a folyam a kimenő ágon, ha minden bemenő folyam beérkezett. Így a példa alapján csak akkor következik a helyszíni összeszerelés, ha mind a váz elkészítése, mind a fiókok elkészítése befejeződött.

A bútorkészítés folyamata látható az alábbi ábrán.



30. ábra: A bútorkészítési tevékenységben látható az elágazás és az illesztés, amely két csomópont között lévő műveletek párhuzamosan futnak.

Két vagy több párhuzamosan futó művelet nem biztos, hogy egyszerre fejeződik be. Az illesztő csomópont feladata, hogy bevárja minden beérkező folyam esetében a műveletek befejezését, és után adja át a vezérlést az illesztő csomópont utáni műveletnek.

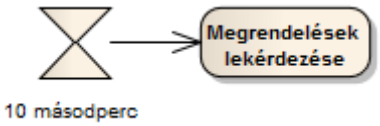
Idő események kezelése

Egyes esetekben a tevékenységek tartalmaznak olyan eseményeket, amelyekre időfüggés jellemző. Lehet ez egy késleltetés, egy tevékenység elkezdése bizonyos időpontban vagy periodikus lefutás. A bútorkészítés folyamatában az igények felmérése és az igények egyeztetése között 3 nap szükséges.

Az idő eseményeket homokóra szimbólummal jelölik. A következő ábra mutatja az időesemény megjelenítését a tevékenységfolyamban. A homokóra mellett látható a várakozási idő időtartama. Az idő esemény aktivitást a bemenő élen beérkező folyam jelzi. A tevékenységfolyamban itt egy várakozás aktiválódik, a jelölt ideig.

31. ábra: Az ábrán megjelenített időesemény 3 napos várakozási időt jelöl a két művelet között.

Ha az időesemény nem rendelkezik bemenő ággal, akkor az egy periódikusan ismétlődő műveletet jelent. A periódikusan ismétlődő művelet gyakorisága fel van tüntetve a homokóra jelölés mellett. Így jelölhető egy weboldal frissítési gyakorisága, vagy a beérkezett megrendelések lekérése.

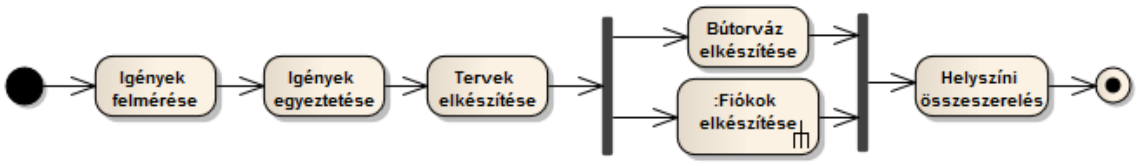


32. ábra: Periódikusan ismétlődő esemény jelölése tevékenységfolyamban.

Az előző ábrán látható módon a tevékenységfolyam nem feltétlenül kezdődik kezdő csomóponttal. A tevékenység elkezdése egy időpont bekövetkezésétől vagy egy periódus lejártakor is lehetséges.

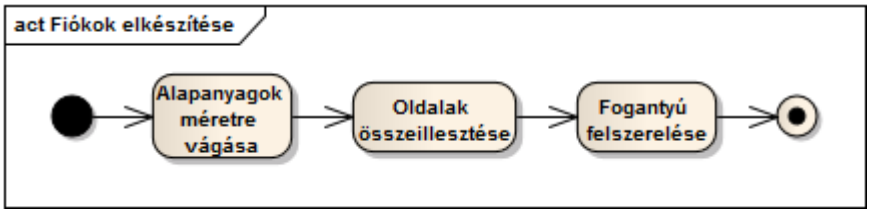
Összetett műveletek, tevékenységek alkalmazása

Ha a modellezett rendszer nagyméretű és a modellezett tevékenységdiagram túl nagy, vagy egy művelet többször is lefut, lehetőség van az olvashatóság növelése érdekében a művelet részletes kifejtésére egy másik diagramban. Ezzel a lépéssel az eredeti, tartalmazó diagram könnyebben átláthatóvá válik. A következő ábra mutatja a bútorkészítés tevékenységének diagramját. A fiókok elkészítése művelet összetett műveletként szerepel, mivel más tevékenységet hív meg. A jelölése lefelé fordított villa szimbólummal történik. Ilyen esetben tehát a művelet egy újabb tevékenységben kerül kifejtésre.



33. ábra: A tevékenységdiagram egyszerűsített formában. Javítja az olvashatóságot a tevékenység meghívása.

Az alábbi ábrán látható a Fiókok elkészítése kifejtett tevékenység.



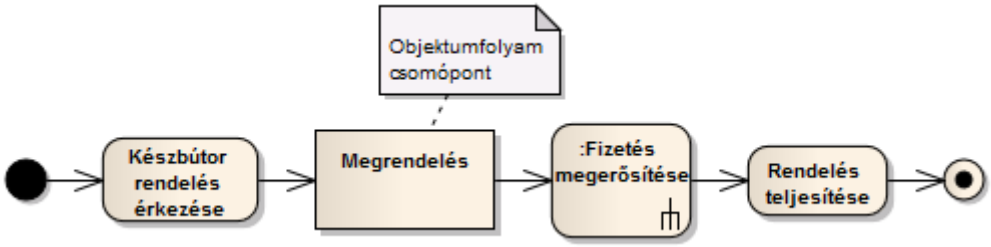
34. ábra: A Fiókok összerakása tevékenység egy folyamatként írható le.

A tevékenységdiagramnak megvan a saját kezdő csomópontja és záró csomópontja. A záró csomópont jelöli a Fiókok összeszerelése tevékenység végét, azonban nem jelenti a teljes folyam végét. Ha a tevékenység befejeződik, a vezérlés átadódik a tartalmazó tevékenységnek, amely ezután a szokásos módon fut le. Az ábrán látható módon a meghívott tevékenység keretben helyezkedik el. Ez segíti a tevékenységek közötti összekötések azonosítását.

Objektumok kezelése

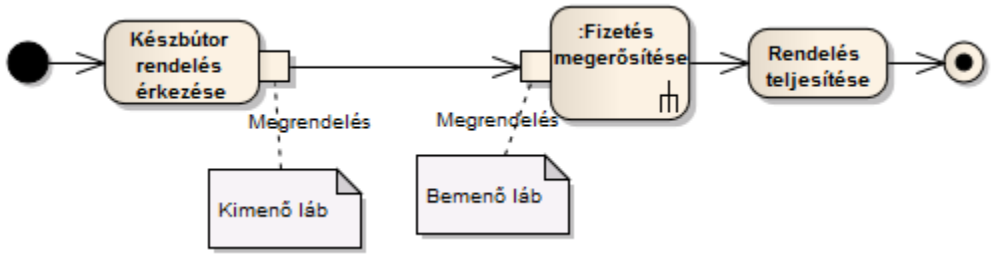
Egyes esetekben a modellezett folyamatokban fontos szerepet játszanak az adatobjektumok. Például a kész bútorok eladásánál a megrendelési folyamatban fontos dokumentumként szerepel maga a megrendelés és annak egyes állapotai. Ez a megrendelés dokumentum tartalmaz minden az egyes lépésekben fontos adatot. A tevékenységdiagramokban az adatok többféleképpen is szerepeltethetők.

A műveletek közötti objektumok objektumfolyam csomópontokként vagy adatfolyam csomópontokként szerepelnek. Az objektum jelenléte a folyamban jelöli, hogy a tevékenység egy adott pontján az adott objektum használatban van, létrehozták vagy módosították a körülötte lévő műveletek által. Az objektumfolyam csomópontok jelölésére téglalapot használnak. Az alábbi ábrán látható megrendelési objektum a Készbútor rendelés beérkezése művelet eredményeként, míg a Fizetés megerősítése művelet bemeneteként szerepel.



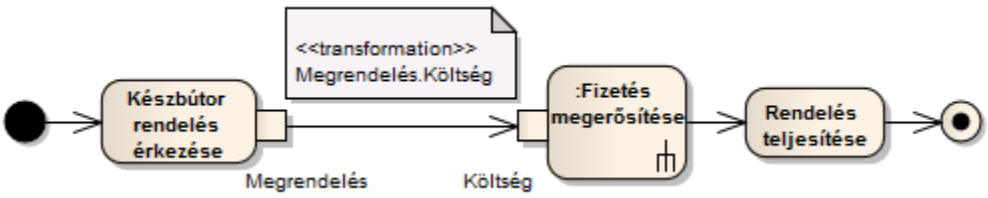
35. ábra: A Megrendelés objektumfolyam csomópont fontos adatot hordoz a Fizetés tevékenység szempontjából.

Ha ennél a jelölésnél egyértelműben szükséges jelölni azt, hogy a dokumentum egy művelet kimenete, illetve egy másik bemenete, érdemes lábakat (pin) használni a műveleteknél. A kimeneti láb jelöli, hogy egy művelet kimeneteként egy dokumentum képződik, a bemenő láb jelenti, hogy a művelet végrehajtásához szükséges egy adott dokumentum. Ennek hiányában a művelet nem hajtódik végre.



36. ábra: Műveletek kimenetének és bemenetének kiemelése lábakkal

Ha a Fizetés végrehajtása művelet elvégzéséhez csak a Megrendelés objektum egy részére van szükség, akkor az UML lehetőséget ad az eredeti objektum transzformációjára. A transzformációval az eredeti dokumentumból kialakítható az a dokumentum, amelyre szükség van. A következő ábra mutatja be, hogy a Fizetés megerősítése műveletnek a Költség objektumra van szüksége a végrehajtáshoz, és ezen Költség objektum egy transzformációval áll elő a megrendelés objektumból.



37. ábra: A transzformáció megmutatja a bemeneti objektumok származását

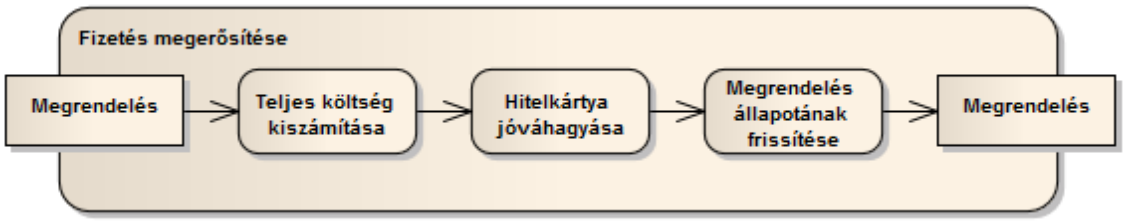
Az objektumok kezelésének egy következő módja az objektumállapotok megjelenítése a folyamban. Az ábrán látható módon a Megrendelés objektum különböző állapotokban szerepel a tevékenység előrehaladása során. Az objektum állapotát az objektum neve alatt elhelyezett szögletes zárójelben

jelölik. A Megrendelés objektum állapota a Fizetés megerősítése művelet előtt függőben lévő állapot, míg a fizetés után jóváhagyott állapotba kerül.

38. ábra: A Megrendelés objektum állapotának változásai a tevékenységben.

Objektumok, mint a tevékenységek bemenetei és kimenetei

A műveletek kimenetei és bemenetei mellett az objektumok tevékenységek bemenetei és kimenetei is lehetnek. A tevékenységek bemeneteiként és kimeneteiként szereplő objektumok a tevékenységek keretén kerülnek megjelenítésre. Ezzel egyértelművé tehető, hogy a teljes tevékenység bemenetet és kimenetet igényel. A következő ábrán látható, hogy a Fizetés megerősítése tevékenység bemente és kimenete a Megrendelés objektum.



39. ábra: Az objektumok használata a tevékenységek bemeneteinek és kimeneteinek kiemelésére

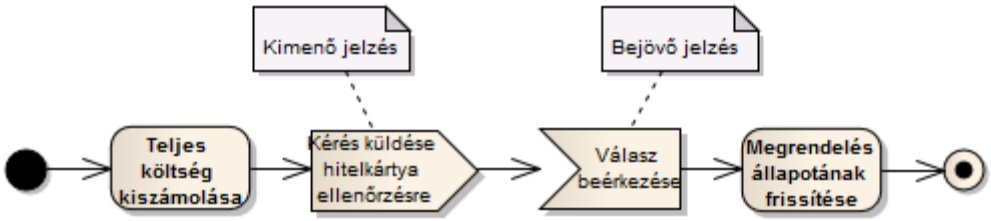
Jelzések küldése és fogadása

A tevékenységek a rendszerben általában nem magukban futnak, hanem interakciót folytatnak a rendszeren kívül lévő folyamatokkal, szereplőkkel, vagy más rendszerekkel. Ezen interakciók kezelésére alkalmazza az UML a jelzéseket (signal). Jelzés lehet, például ha a fizetési folyamatban a Fizetés megerősítéséhez szükség van a hitelkártya kiadó bank jóváhagyására. A tevékenységekben jelzések kimenő vagy bejövő jelzések lehetnek. Erre példák a következők:

- A rendszer kimenő jelzést küld a hitelkártyát kibocsájtó banknak, akitől a későbbiekben bejövő jelzésként egy visszaigazolás érkezik.
- A megrendelésről érkező kérés indikálja a megrendelés tevékenység elindulását. Ilyen esetben a kérés, mint bejövő jelzés jelenik meg.
- A billentyűzeten egy gomb megnyomása a gombnyomáshoz kötött kód elindulását vonja maga után. Ez egy bejövő jelzést jelent a rendszerben.
- A rendszer által küldött jelzés a megrendelés elindításáról a megrendelőnek kimeneti jelzésként jelentkezik a rendszerben.

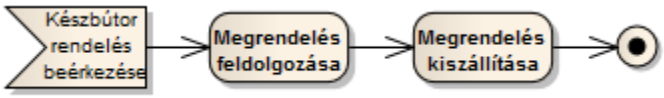
A bejövő jelzés egy művelet végrehajtását indikálja a tevékenységdiagramban. A bemeneti jelzésre váró művelet ismeri a jelzésre való reagálás módját, és arra vár, hogy a jelzés megérkezzen. Az azonban nincs egyértelműen leírva, hogy a jelzés pontosan mikor érkezik meg. A kimenő jelzések egy a rendszerből kifelé, külső partnernek küldött jelek, üzenetek. Ezen jelzések kiváltanak valamilyen reakciót a partnerből, ez azonban a modellben nem jelenik meg.

Az alábbi ábra mutatja, hogy a bankkártya jóváhagyásához szükség van a bankkártyát kibocsájtó bankkal történő kommunikációra. Ez egy kimenő jelzésként jelenik meg a tevékenységben. A jelzés kiküldése után a vezérlőjel nem áll meg az adott műveletnél, hanem továbblép a következő műveletre. A bejövő jelzés külső rendszerből érkezik. A bankkártya kibocsájtó banktól érkező bejövő jelzés visszaigazolást ad a bankkártya elfogadásáról. Az ehhez kapcsolódó művelet várja a bejövő jelzést, és akkor hajtódik végre, ha a jel megérkezett.



40. ábra: Bejövő és kimenő jelzések a hitelkártya elleőrzésére

Ha a tevékenység egy bejövő szignállal kezdődik, akkor az adott csomópont folyamatosan várakozó állapotban van, és a megfelelő jelzés beérkezésére vár. A következő ábrán látható példa szerint a tevékenység mindannyiszor lefut, ahányszor megrendelés érkezik.



41. ábra: Tevékenység indítása beérkező jelzés használatával.

Tevékenységek indításának lehetőségei

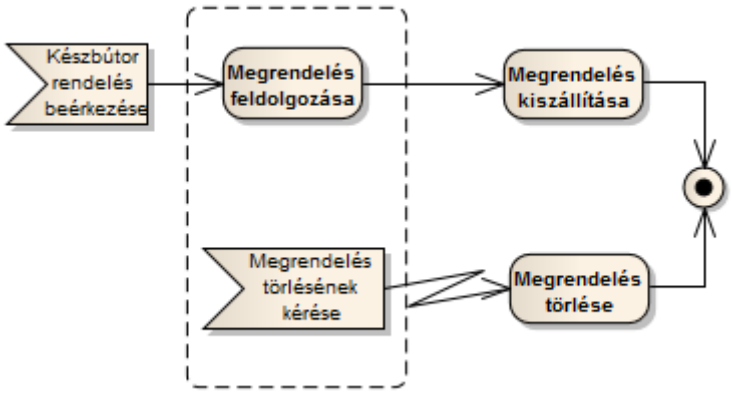
A legegyszerűbben elkezdhető egy tevékenységdiagram a szokásos kezdő csomópont használatával. Emellett azonban – ahogyan az előző példán is láttuk – több lehetőség is rendelkezésre áll a tevékenységfolyamok elindítására:

- a tevékenység elkezdődhet egy adat beérkezésével
- egy idő esemény bekövetkezésével
- beérkező jel hatására

Tevékenységek és folyamok lezárásának lehetőségei

Eddig a fejezetben csak egyszerű tevékenység záró csomópontokkal találkoztunk. A tevékenységek kezdéséhez hasonlóan több féle lehetőség áll fenn a tevékenységek zárására is.

Az általános lefutású tevékenységeknél minden művelet befejezi a futását és átadja a vezérlést a következő műveletnek, majd ha a tevékenység a végéhez ér, befejeződik. A valóságban azonban nem minden művelet fut le valamilyen beavatkozás, megszakítás nélkül. Egy szoftverrendszer futását a felhasználó több esetben is megszakíthatja. a megszakítások a folyamatban bekövetkező jelzések, események hatására következnek be. Adott megszakítás általában a futás közben csak bizonyos pontokon megengedett. Ez azt jelenti, hogy csak a folyamat egyes műveleteinél értelmezhető egy megszakítási esemény beérkezése. Ennek a területnek a jelölésére használatos a megszakítási régió. A megszakítási régiót szaggatott vonallal jelölik és keretbe foglalja a megszakított műveletet vagy műveleteket és a megszakítást indikáló eseményt. A megszakítási esemény kimente egy villám alakú nyílban jelenik meg.

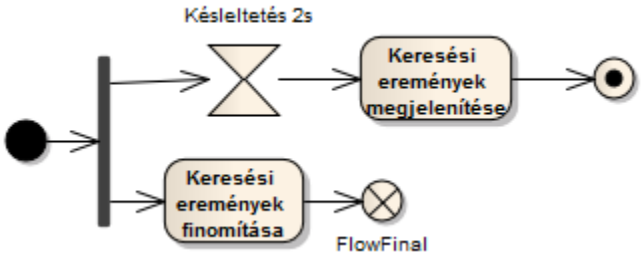


42. ábra: A megszakítási régiók a tevékenységek lezárásának egyik lehetőségét jelentik

Az ábrán kifejezet módon a felhasználó csak abban az esetben törölheti a megrendelését, tehát szakíthatja meg a megrendelés tevékenységét, ha a megszakítás a Megrendelés feldolgozása művelet végrehajtása közben érkezik. Természetesen a megrendelés nem szakítható meg a megrendelt termékek kiszállítása közben. A Megrendelés törlése esemény beérkezésekor, a vezérlés átkerül a Megrendelés törlése művelethez, és ezen az ágon fejeződik be a tevékenység.

Folyamok zárása

Az UML 2.0 új hozománya, hogy a folyamok anélkül lezárhatóak, hogy az egész tevékenység futása lezárulna. A folyam záró csomópont a csak az adott folyamot zárja le, más folyamban lévő műveletek futására nincs hatással.



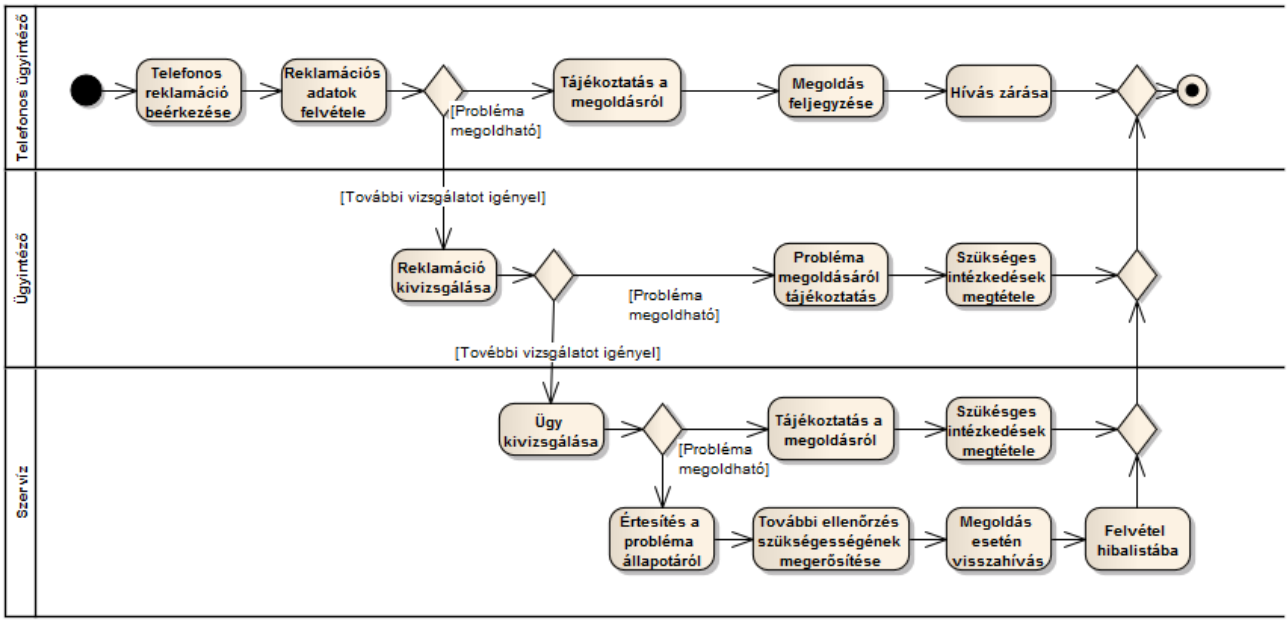
43. ábra: A folyam záró csomópont csak az adott folyam futására van hatással

A fenti ábra egy keresőmotor működését mutatja be. A keresőmotor két eredményablakkal rendelkezik a lehető legjobb eredmények megadása érdekében. A fenti folyamban egy 2 másodperces késleltetés látható. A késleltetés alatt a lenti folyamban a keresés finomítására van lehetőség. Ha finomítás eredményesen tér vissza, akkor az adott folyam lezárul. Ha a finomítás nem fejeződik be a fenti folyam futásának ideje alatt, akkor a keresés eredményei finomítás nélkül kerülnek megjelenítésre.

Résztvevők megjelenítése a tevékenységekben

A tevékenységekben különböző résztvevők vehetnek részt. Ezek a résztvevők lehetnek konkrét személyek, vagy bizonyos feladatkörrel rendelkező munkatársak, csoportok. Ha egy tevékenység modellezésekor fontos szerepeltetnünk a résztvevőket, azt partíciók alkalmazásával tehetjük meg. A bútor kiszállításhoz kapcsolódóan szükség van egy a kiszállítást intéző részlegre, és a számlázási részegre is. Egy másik példában a szoftverek támogatását végző többszintű részlegre bontható a folyamat.

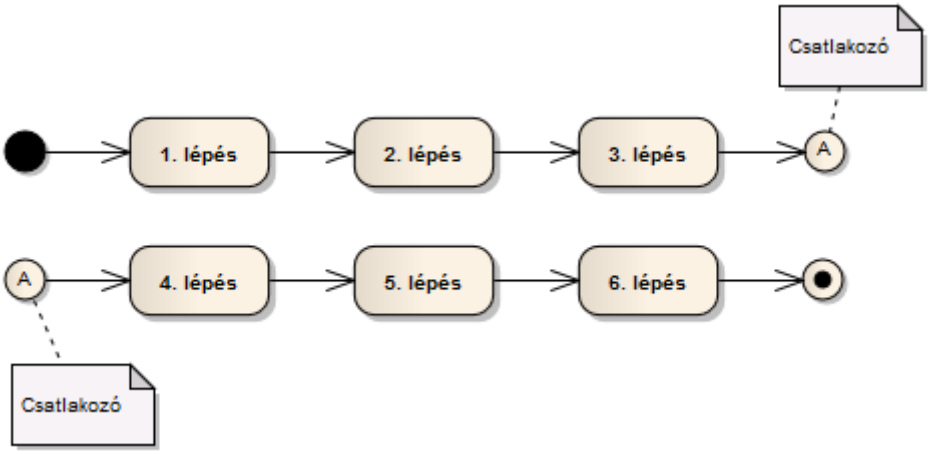
A partíciók használhatóak annak a jelzésére, hogy az egyes műveleteket mely szereplők végzik. Ilyenkor a tevékenységfolyam egy átalakítására van szükség, hogy a folyamat jól olvasható legyen. A partyciók hazsnálatára jó példa a korábban említett üzleti folyamat diagram ahol sz úgynevezett swimlane-ek illetve pool-ok alkalmazása szinte elengedhetetlen az üzleti folyamatokban résztvevő személyek azonosítására.



44. ábra: Partíciók megjelenítése a tevékenységdiagramban a szereplők azonosításához.

Nagyméretű diagramok kezelése

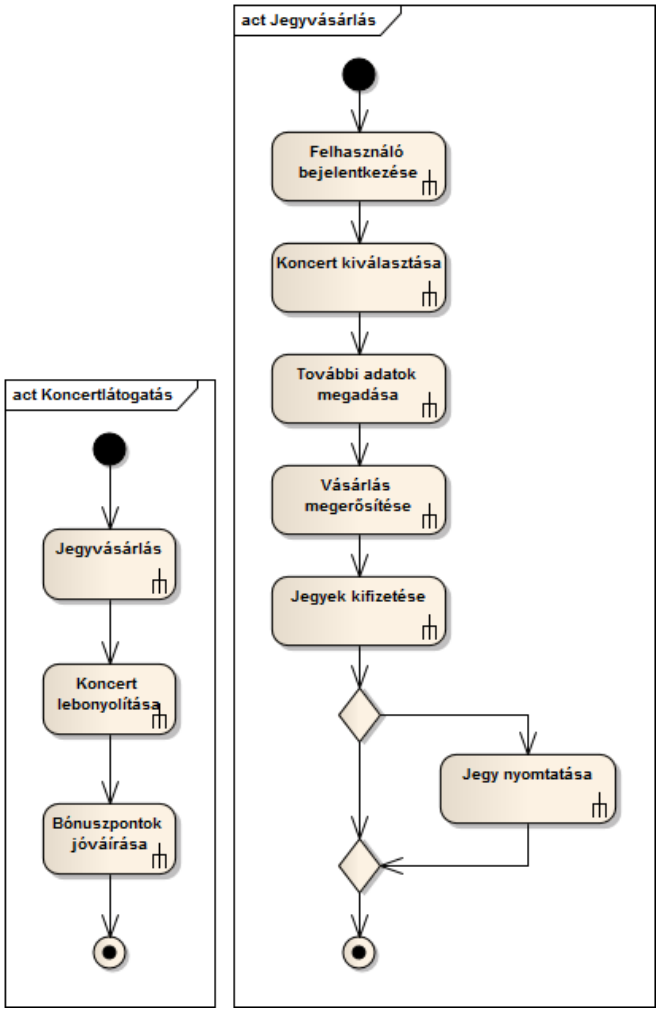
A rendszerek többségének modellezéséhez az eddig tárgyaltakon kívül jóval többféle jelölési lehetőségre szükség lenne. Ezeket a jelöléseket az UML magában foglalja, de ezen tárgy keretein belül nem tárgyaljuk. A nagyobb rendszerek nagy valószínűséggel nagyobb méretű és komplexebb tevékenységdiagramok leírására kényszerítik a modellezőt. Ahhoz hogy a diagramok átláthatóak és értelmezhetőek legyenek bizonyos kisegítő jelölésekre van szükség. Egyik ilyen lehetőség a csatlakozók vagy kapcsolódási pontok alkalmazása. A csatlakozók feladata, hogy az egymáson esetleg többszörösen is átmenő vonalakat, éleket megszüntessék a diagramban anélkül, hogy a diagram értelme változna, viszont ezzel javítsák az olvashatóságot. A csatlakozók jelölésére számokkal vagy betűkkel ellátott köröket használnak. Ugyanazzal az írásjellel ellátott csatlakozó kapcsolódik a folyam egyik, illetve másik végéhez, mutatva ezzel, hogy a folyam egyikből a másikba folytatódik. Ebből következően a párból az egyik csatlakozó csak bemenő, míg az másik csak kimenő éllel rendelkezik. A csatlakozók alkalmazásánál is tartani kell a mértéket: a túl sok csatakozó használata szintén csökkenti az olvashatóságot.



45. ábra: A csatlakozók jobban olvashatóvá teszik a nagyméretű diagramokat.

5. Tevékenységdiagramok

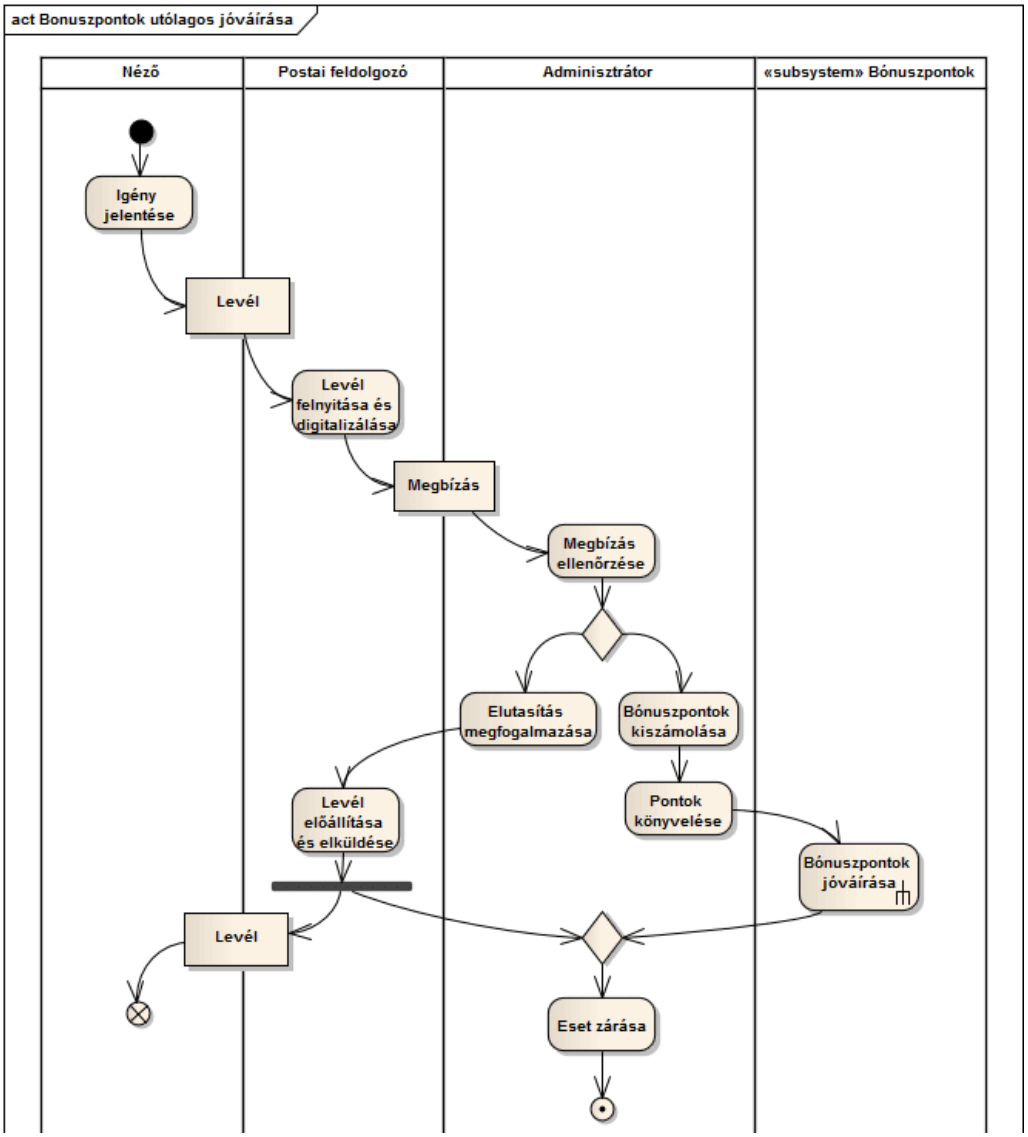
A tevékenységdiagramok tevékenységekből, és a köztük kapcsolatot teremtő vezérlési folyamélekből állnak. A mintarendszerre vonatkozó egy tevékenységdiagram látható az alábbi ábrán.



46. ábra: Koncertlátogatás és Jegyvásárlás folyamatok tevékenységei

Az összetett tevékenységeket, amelyek további tevékenységlépések folyamából állnak össze egy lefelé fordított villával jelezzük. A Jegyvásárlás tevékenységdiagramban láthatóak az UML-ben már megszokott esetválasztó és egyesítő csomópontok, rombusszal jelölve. Így két lehetőség közül választhatunk: elektronikus jegy nyomtatása, vagy a folyamat a másik vezérlőfolyamon megy tovább és a végállapotba kerül.

A tevékenységdiagramoknál megtehetjük, hogy partíciókat vezetünk be, amellyel azt modellezzük, hogy az adott tevékenységet melyik aktor végzi, melyik aktor kapcsolódik hozzá.

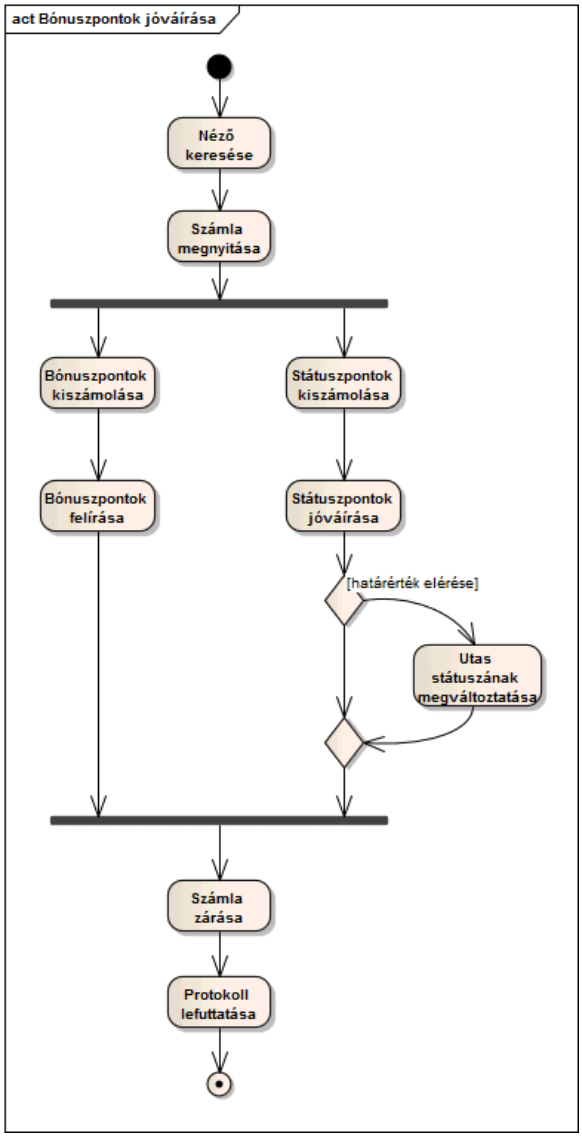


47. ábra: Bónuszpontok utólagos jóváírása

Az ábrán a bónuszpontok utólagos jóváírása látható. Megjelennek a folyamatban résztvevő objektumfolyam-csomópontok - mint például a levél és a megbízás - amelyek a megvalósítás során megfelelő osztályokra utalnak. A levélküldés vezérlési folyamata egy folyamvég pontban végződik, amely vezérlőjeleket fogad, de egyéb kihatásuk nincs a folyamatra. A végállapot ezzel szembe az egész folyamat végét jelenti. A bemutatott tevékenységgel eddig a leírások során még nem találkoztunk. A modellünk tehát kiegészítésre került újabb adatokkal, amelyeket a használati esetek és az üzleti folyamatok szintjére is vissza kell vezetnünk, tehát az itt megjelenő folyamatnak is készítenünk kell táblázatot és döntés alapján a használati eset vagy az üzleti folyamat leltárban is meg kell jelennie. Ugyanígy az újonnan megjelent aktorainkat is ábrázolni kell.

Használati esetek lefutása

Az utólagos bónuszpont jóváírás üzleti folyamat lefutásának egyes lépései mind használati esetek. A Bónuszpontok jóváírása funkcionalitást a BonusPoints alrendszer valósítja meg. Ez a használati eset részletezhető egy tevékenységdiagrammal.

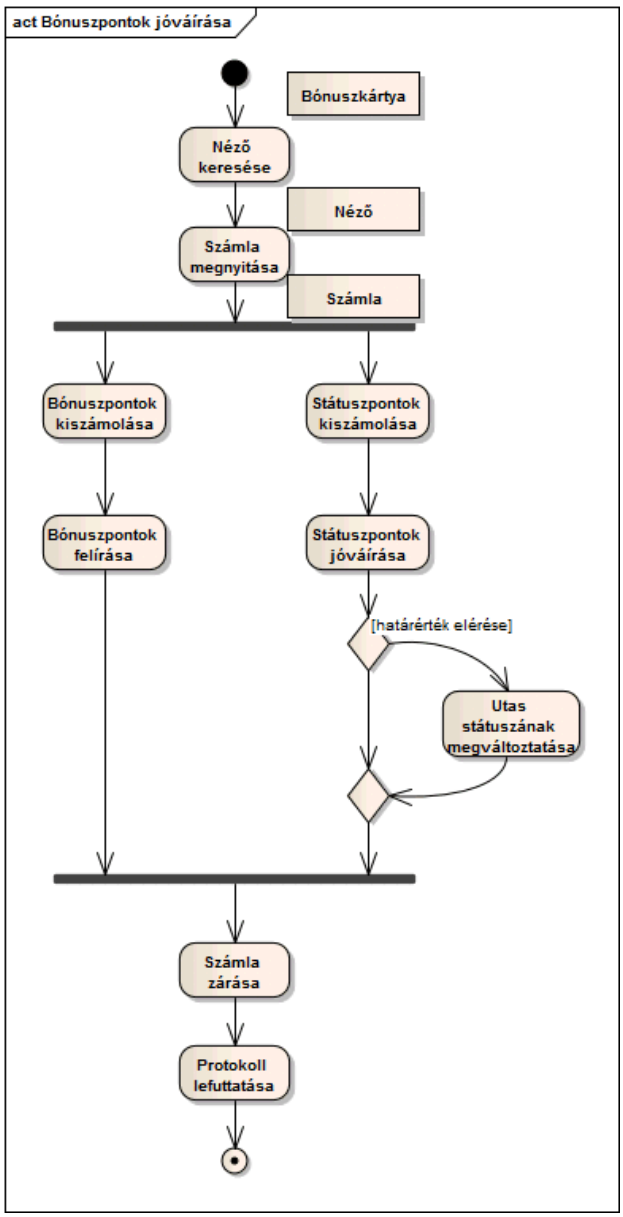


48. ábra: A bónuszpontok jóváírása használati eset lefutása

Látható, hogy az Néző azonosítása után a tevékenységfolyam egy elválasztó csomóponttal szétválasztva, két szálon folyik tovább. Mind a két szál eseményei lefutnak, majd ha mind a bónuszpontok jóváírása, mind a státusz kiszámolása megtörténik, az összeolvasztó csomóponttal jelezhető, hogy egy szálon folyik tovább. A két vezérlési szál tehát egyszerre fut, és csak akkor folytatódik tovább a közös szálon a tevékenység, ha mindkettő befejeződött.

A korábbi UML verziókban az elválasztás és az összeolvasztás csomópontoknak zárójelstruktúrában párosan kellett megjelenniük, ez az UML 2-től kezdve már nem kell, hogy így legyen.

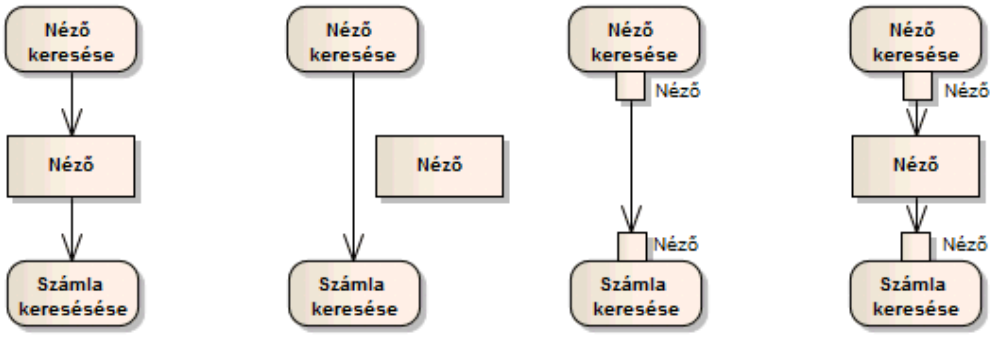
A tevékenységdiagramon az adatfolyamot is megjeleníthetjük. Az előbbi példán látott tevékenységdiagram adatfolyam csomópontokkal kiegészítve a következőképpen ábrázolható:



49. ábra: Bónuszpontok jóváírása adatfolyamokkal

Az adatfolyam csomópontok jelölésére három alternatív lehetőség létezik: „szabadállású”, – ilyenkor a tevékenységeket és az objektumfolyam-csomópontokat adatfolyam élt használnak - a „kiegészített”, – az objektumfolyam-csomópont a vezérlési folyamél mentén helyezkedik el, későbbi kiegészítése könnyen lehetséges, - valamint a csatlakozólábas jelölés – annak hangsúlyozására, hogy az adatok

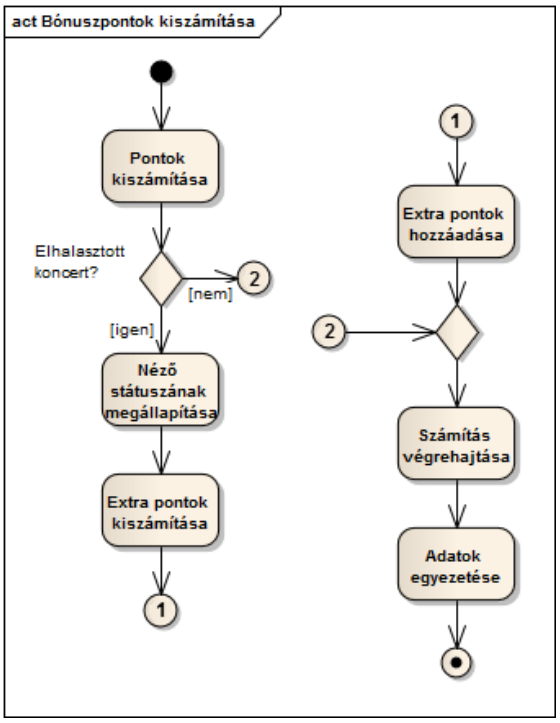
valamely tevékenység kimenetei vagy valamely tevékenység bemenetei, és ezért paraméterként szerepeltethetőek.



50. ábra: Adatfolyam egyenértékű jelölései: kiegészített, szabadállású, csatlakozólábas és ezek kombinációja

Csatlakozók

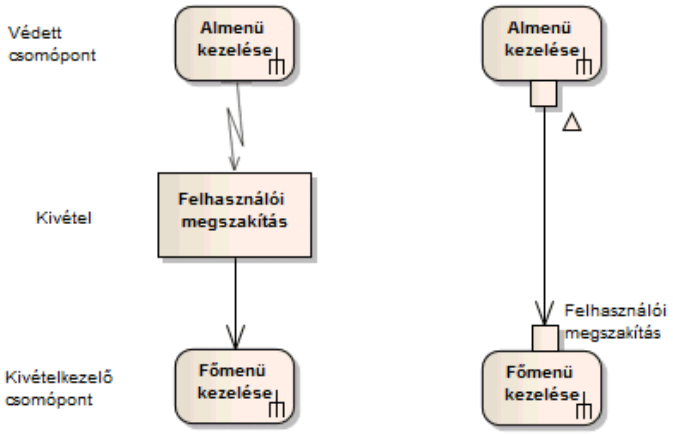
Előfordul, hogy egy tevékenységfolyam túl hosszú, túl bonyolult lesz, nem lehet a folyamot egyszerűen kezelni. Ilyen esetben használhatóak az ugrópontok, amelyek azt szimbolizálják, hogy egy folyam hol folytatódik. Jelölésbeli összekötőről van tehát szó. Egy diagramban ez a jelölés a következőképen néz ki:



51. ábra: Ugrójelek alkalmazása átfedő tevékenységfolyamok esetén

Kivételek

A kivételek kezelésére olyan esetben van szükség, ha pl. egy hiba hatására a tevékenységet meg kell szakítani, és a hiba kezelése a tevékenységen kívül folytatódik tovább. Ennek jelölése az ábrán látható módon történik.

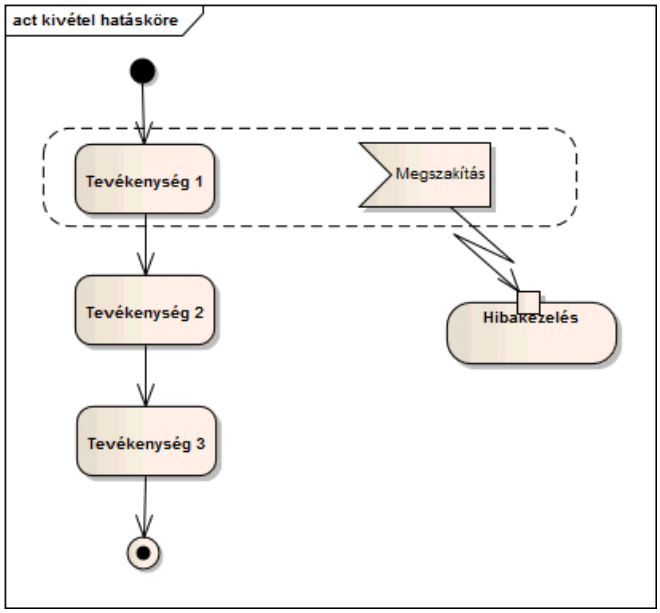


52. ábra: Kivétel kiváltása védett csomópontban és kezelése kivételkezelő csomópontban

A kivétel a védett csomópontban váltódik ki – az ábrán az Almenü kezelése – és a kivételt a Kivételkezelő csomópont fogja lekezelni – Főmenü kezelése. Egy kivételkezelő csomópont szintén válthat ki kivételeket, így egész láncolat jöhet létre. Tevékenység: egy általánosan ismert tevékenység váltja ki az eseményt

Megszakítási régiók

Alapesetben egy kivétel egy tevékenységre hat és ez a tevékenység be is fejeződik. Ha meg szeretnénk határozni egy területet, amin belül a megszakításnak hatása van, azt a megszakítási területek alkalmazásával tehetjük meg. Ilyen esetben a kivétel azon a területen belül lévő tevékenységekre lesz érvényes, és ha a többi tevékenység futásánál váltódik ki a kivétel, azokra nem lesz hatással. A megszakítási terület alkalmazásának lehetőségét mutatja az ábra.



53. ábra: Kivétel kiváltása a megszakítási területen belül szakítja csak meg a vezérlőfolyamot.

6. A rendszer strukturális modellezésének lépései osztálydiagrammal

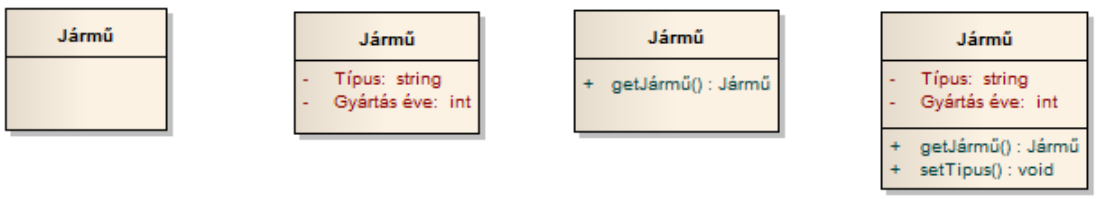
Az objektum-orientált szoftverfejlesztés egyik kulcsfogalma az osztály. Segítségével megvalósítható többek közt az egységbezárás, amely adatok és rajtuk végzett műveletek szoros kapcsolatát jelenti. Az osztály feladata az, hogy leírja a tekintett fogalom főbb tulajdonságait, valamint tartalmazza azon műveleteket is, amelyekkel üzeneteket küldhetünk az osztályból létrehozott objektumpéldányoknak. Egy példával illusztrálva az elmondottakat osztályként tekinthetünk a különféle járművekre, amelyeket jellemez például a típus, a szín, a gyártás éve és még hosszan sorolhatnánk. Ezen jármű osztályból létrehozhatunk konkrét jármű egyedeket, amelyek a valóságban is léteznek. Mondhatjuk például azt, hogy egy 2014-ben gyártott fehér Ford Focus gépjárművet valósítunk meg és írunk le.

Modellezés során is lehetőségünk van UML segítségével leírni általában egy osztályt és az abból létrehozott objektumpéldányokat. Előbbire az osztálydiagram, utóbbira az objektumdiagram ad lehetőséget.

Az osztálydiagram tulajdonképpen egy (gyakorlatilag összefüggő) gráfként fogható fel, ahol a gráf csomópontjai az osztályoknak feleltethetők meg, a gráf élei pedig az osztályok közötti kapcsolatokat, relációkat mutatják. Egy osztályt górcső alá véve alapvetően három rész különíthető el:

- Minden osztálynak adunk nevet
- Felsorolhatjuk az osztályt jellemző tulajdonságokat (programozói terminológiát használva adattagokat)
- Valamint megadhatjuk azokat a metódusokat, amelyekkel kommunikálhatunk az osztállyal.

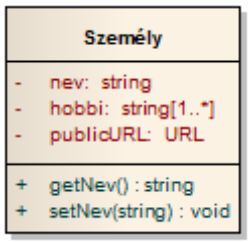
Az alábbi ábra az osztályok megjelenésének négy különböző módját mutatja. Egy osztályt szemléltethetünk felső nézetben csupán a nevével megadva, de alkalmazhatunk olyan osztályreprezentációt is, ahol feltüntetjük az adattagokat és/vagy metódusokat is.



54. ábra: Osztályok különböző részletességű leírása

Megjegyezzük, hogy mindegyik reprezentációnak megvan a helye a szoftvertervezés folyamatában. A tervezési fázis elején egy felső szintű osztályazonosítási folyamat megy végbe, majd fokozatos finomítás eredményeképpen kapjuk meg elsőként az adattagokat, majd később a metódusokat.

A lenti ábra egy **Személy** nevű osztályt mutat be. Mint látható itt az osztályban három rész különíthető el: a felső harmad az osztály nevét, a második harmad a tulajdonság megnevezését, míg az utolsó harmad a metódusok felsorolását tartalmazza.



55. ábra: Egy egyszerű osztály grafikus reprezentációja

Mind az adattagok, mind a metódusok nevei előtt megadjuk, hogy a tekintett adattag vagy metódus milyen láthatósággal rendelkezik.

Hozzáférési szintek, láthatóság

Alapvetően négy hozzáférési szintet definiálhatunk:

- Publikus hozzáférést (**public**), amely adattagok esetén teljesen szabad hozzáférést és módosítást engedélyez az adott változóhoz a programunk bármely pontjáról. Metódusok esetén is ez a legmegengedőbb hozzáférési szint, hiszen az ilyen címkével ellátott függvényeket is szabadon meghívhatjuk. Az UML szintaktika egy "+" jellel reprezentálja a publikus hozzáférési szintet. Általában elmondható, hogy a legtöbb metódust publikusként hozzuk létre, biztosítva így azok meghívhatóságát az osztályon kívülről is.

- Már szigorúbb hozzáférési szintnek tekintjük a **protected**, azaz védett hozzáférést. Az objektum-orientáltság egyik legfontosabb tulajdonságakor, az öröklötetésnél van fontos szerepe. Az ős (szülő) és gyerekosztály kapcsolatakor tudjuk szabályozni az ősosztályban megtalálható adattagok és metódusok elérhetőségét és láthatóságát a gyermekosztályokban. A védett hozzáférés tehát azt szolgálja, hogy minden gyerekosztály elérhesse a protectedként definiált elemeket, míg más osztályok, amelyek nem részesei a szülő-gyermek kapcsolatnak, ne férjenek hozzá az így definiált elemekhez. Az UML diagram # segítségével ad módot arra, hogy ezt grafikusan is szemléltessük a megrajzolt diagrammokon.

- A csomag szintű hozzáférési szintről (**package** hozzáférés) is szót kell ejtenünk. Osztályainkat un. csomagokba sorolhatjuk, amely csomagon belül található osztályok metódusai szabadon elérhetik egymást. Biztosíthatjuk így azt, hogy az ős-gyermek tulajdonsággal nem rendelkező osztályok között fennállhasson a hozzáférés korlátozása. Nevezetesen, a csomagon belüli osztályok számára ez egy public hozzáférési szintet jelent, míg minden egyéb osztály nem hozzáférhetőként látja a csomag szintű hozzáféréssel megcímkézett elemeket. Ez a nem hozzáférhető kapcsolat less a private szint, amelyről a következő pont tesz említést. A csomag szintű hozzáférést ~ jel mutatja az osztálydiagramban.

- A private láthatóság a legszigorúbb eleme a hozzáférés szabályozásnak. Csak az adott osztályon belül érhetjük el az így megjelölt adattagokat és metódusokat, kívülről ezek elérése nem lehetséges. Az objektum-orientált filozófiát követve általában igaz, hogy adattagjainkat

ilyen hozzáférésűre állítjuk, biztosítva így az adatelrejtés lehetőségét. Jelölésére az UML diagram a “-“ jelet alkalmazza.

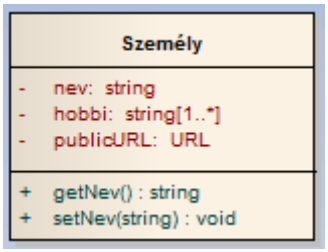
C++ nyelven az ismertett Személy osztály a következőképpen definiálható:

```
class Szemely {
      string nev;
public:
      void setNev(string);
      string getNev();
}
```

Két hozzáférési szint fedezető itt fel: a private, amely alapértelmezett a nyelvben, valamint a public, amelyet jeleznünk kell a kódban. Megjegyezzük, hogy C++-ban csomag hozzáférési szint definiálására nincs mód kulcsszó használatával.

Attribútumok

Az osztályt jellemző tulajdonságokat attribútumokkal (adattagokkal) írjuk le. Mint azt láttuk fent megadjuk az adott attribútum hozzáférési szintjét, annak nevét, majd típusát. Érdemes gondot fordítanunk a tulajdonságot leíró adattag nevének gondos megválasztására, hiszen az a későbbiekben segít bennünket a program áttekintésében és megértésében. Természetesen egy adattag adatok kollekciójaként is viselkedhet, ebben az esetben ezt az osztálydiagramon külön jelezzük. Ez azt mutatja, hogy egy változó több azonos típusú adat tárolására is képes.

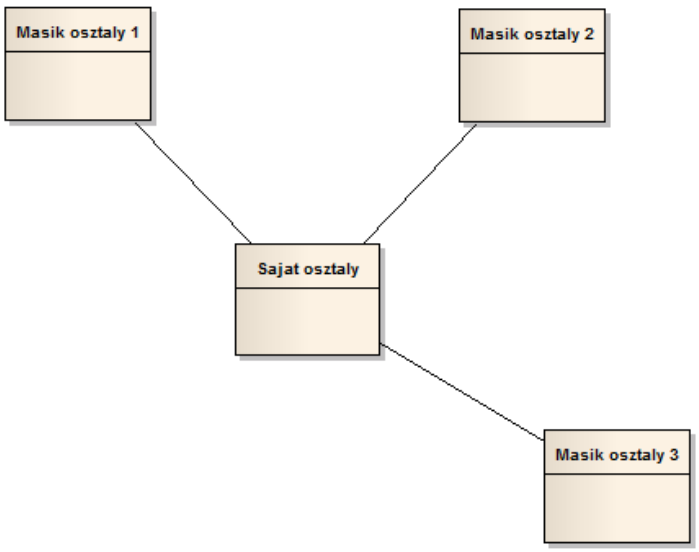


56. ábra: Osztály grafikus reprezentálása UML-ben

A fenti ábrán a Személyekhez több hobbitevékenység is tartozhat, amelyet ez esetben stringek segítségével írunk le. Az itt megadott * karakter jelzi azt, hogy e tevékenységek száma nincsen korlátozva, abból sok előfordulást tudunk tárolni. A publicURL adattag azt az URL címet tárolja, amely a tekintett személy profiloldalára mutat. Hangsúlyoznunk kell itt ezen attribútum típusát, amely URL, nem szabványos primitív adattípus. E típus egyúttal azt is előrevetíti, hogy a diagramunk valamely más részén ezt megtaláljuk osztályként, ahol a típus pontos részleteit meg fogjuk ismerni. Megemlítjük azt

is, hogy az attribútumok további tulajdonságai is feltüntethetőek az osztálydiagramon. Ilyen tulajdonság lehet például az, ha egy adattag konstans, vagy akár az is, ha egy kollekcióban tárolt adatok sorrendjét rögzíteni akarjuk. Az előbbi esetben gyakran a „readOnly”, míg utábbi esetben az „ordered” kulcsszót tüntetjük fel a változó típusa mellett. Számos egyéb jelző adható még meg, ezen kulcsszavak listájáról az UML specifikáció ad részletes tájékoztatást.

Bizonyára az olvasóban is felmerült a kérdés a publicURL adattag kapcsán, hogy miként is célszerű azt szemléltetni, hogy egy adattag olyan általunk létrehozott típusú, amely az osztálydiagramunk más pontján fellelhető. Jelen kérdés elvezet bennünket az asszociációk fogalmához, amely egy módja annak, hogy szemléltessük az osztályaink közötti kapcsolatokat.



57. ábra: Osztályok és asszociációs kapcsolataik

Két vagy több osztály objektumainak „valamilyen” relációval történő összekapcsolását értjük asszociácó alatt. A fenti ábrát megnézve ez azt jelenti, hogy a „Sajat osztaly” kapcsolatban van a „Masik Osztaly 1”, „Masik Osztaly 2” és „Masik Osztaly 3” típusokkal (osztályokkal). Ez természetesen máshogy is kifejezhetjük:

58. ábra: Osztály típusú adattagok

A fenti két osztálydiagram alapvetően ugyanazt a struktúrát írja le: definiál egy Sajat osztályt, amely aztán három általunk definiált osztállyal van kapcsolatban. Útmutatásként a következő szabályt célszerű betartanunk a jövőben: egy osztály adattagjaként általában a rendszer beépített típusait tüntetjük fel (int, double, string, …), míg az általunk létrehozott típusokra asszociációval hivatkozunk. Ez növeli az áttekinthetőséget és nem utolsó sorban a program egyes részeinek újrafelhasználását is segíti.

Metódusok

Az objektumok közötti kommunikációt vagy más néven üzenetküldést a metódusok valósítják meg. Egy osztálydiagramot reprezentáló téglalap utolsó harmadában kapnak helyett e deklarációk. Megadjuk az adott metódus láthatóságát, nevét, paraméterlistáját, valamint visszatérési típusát. Erre láthatunk példát az alábbi osztály esetén. Miután egy közösségi oldalnál az adminisztrátor számára igények sora érkezhet, ezért célszerű egy olyan tárolóosztályt is deklarálnunk, amely ezen beérkező kéréseket tárolja. Ahogy az megfigyelhető egy tömb tárolja ezen igényeket és azok számát. Metódusok segítségével ezen kérések kezelhetőek: új felvételére van lehetőség, és már meglévő törlését támogatja a rendszer. Ez utóbbi metódus esetén egy indexet várunk paraméterként, amely egyértelműen azonosítja a törlendő bejegyzést.

59. ábra: Metódusok jelölése osztálydiagramon

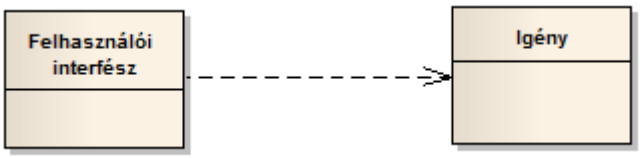
Osztályok közötti kapcsolatok

Mint arról már röviden szóltunk az osztályok között kapcsolatokat definiálhatunk, így az osztályokból létrehozott objektumok üzeneteket küldhetnek egymásnak, azaz egymás metódusait hívhatják meg. Az UML alapvetően öt különböző módot ad az osztályok közötti kapcsolatok típusainak megadására. Ezek a leggyengébb kapcsolattól indulva az egyre erősebb felé a következők:

- Függőség
- Asszociáció
- Aggregáció
- Kompozíció
- Öröklődés

A függőség

A függőség két osztály között azt mutatja, hogy a résztvevő egyik osztálynak ismernie kell a másik osztályt annak érdekében, hogy abból objektum jöhessen létre és az használható legyen. Az ilyen kapcsolatot az UML diagram szaggatott vonallal jelzi és egyúttal irányt is társít a kapcsolathoz. Egy nagyon egyszerű példával szemléltetve az ilyen jellegű kapcsolatot gondolhatunk egy olyan felhasználói interfészre, amely segítségével rögzíthetjük a felmerült igényeket, vagy éppen meg akarjuk azt jeleníteni. Ha ez a felhasználói interfész nem áll rendelkezésre, akkor igény sem hozható létre / jeleníthető meg.

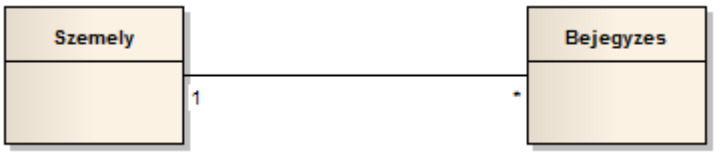


60. ábra: Függőség jelölése

Egy más esetben gondolhatunk függőségre abban az esetben is, amikor például JAVA nyelven a java.math csomag által nyújtott funkciókat akarjuk használni. Ahhoz, hogy egy valós értéket a round függvénnyel kerekítsünk szükség van e csomag használatára. Ilyen szituációk során is függőségi kapcsolat áll fenn, amelyet a diagramunkon reprezentálhatunk.

Asszociációs kapcsolat

Az egyik leggyakrabban alkalmazott és legáltalánosabb kapcsolat két osztály között az asszociáció. Célja kifejezni azt, hogy a tekintett két vagy több osztály valamilyen kapcsolatban van egymással. Példaként említhetjük az adott személyt leíró osztályunkat és a hozzá kapcsolódó közösségi oldalon tett bejegyzéseit, hiszen a személyt köthetjük az általa tett bejegyzésekhez, és azt is tudjuk, hogy egy bejegyzés kihez tartozik. UML jelöléssel a kövtekezőképpen írhatjuk ezt le:



61. ábra: Asszociációs kapcsolat két osztály között

Fontos megemlítenünk a multiplicitások szerepét is a diagrammokon. Ennek oka az, hogy egy objektum valamely osztály több objektumával is kapcsolatban lehet. A fenti kiragadott példát megvizsgálva tüzetesebben szintén elmondható, hogy egy személy több bejegyzést tehet, így ezt a diagramon is meg kell jelenítenünk. Erre szolgálnak a különféle multiplicitás jelölések. Két osztály kapcsolatakor az őket

összekötő élen adhatjuk ezt meg: az él két végére írt érték jelzi ezt a számosságot, mely lehet egy konkrét pozitív egész szám, de lehet a *-gal jelzett karakter is. A konkrét érték, ahogy azt várjuk, azt jelzi, hogy az adott objektum hány más objektummal van kapcsolatban, míg a * tetszőleges értéket jelent 0 és plusz végtelen között. A fenti diagram tehát azt jelöli, hogy egy személy több bejegyzéssel rendelkezhet, míg egy bejegyzés pontosan egy személyhez köthető. A kapcsolat további – értelmezést könnyítő – jelölésekkel is ellátható. Ezekről az UML specifikációban találunk részletesebb leírást.

Aggregáció

Erősebb kapcsolatot jelent két osztály között az aggregációs kapcsolat. Nem csupán általános kapcsolatot akarunk aggregációval kifejezni, hanem a rész-egész viszony fennállását is hangsúlyozni akarjuk. Aggregációs kapcsolatra tipikus példa a konténerosztályban tárolt objektumok esete. Az igényeket egy tömbben, vagy valamilyen több objektum tárolására alkalmas adatszerkezetben kell elhelyeznünk. Így kézenfekvőnek tűnik egy Igények nevű osztály létrehozása, amely konkrét igényeket tud majd tárolni. Egy igény létezhet a programunkban annélkül, hogy azt beletettük volna a futás egy pontján ebbe a tárolóosztlyba, így tehát a két osztály között nem fizikai, csupán logikai tartalmazás áll fent.

Életciklus szempontjából is érdemes vizsgálódnunk a két osztályba tartalmazó objektumok kapcsán, hiszen ez is segít nekünk annak azonosításában, hogy két osztály között aggregációs kapcsolatot érdemes-e felrajzolnunk vagy esetleg valami mást. Ha azt találjuk, hogy a két osztály objektumai nem feltétlenül azonos időpillanatban jönnek létre és szűnnek meg, akkor mérlegelnünk kell az asszociáció és az aggregáció között. E kérdés eldöntésében pedig a rész-egész viszony vizsgálata lesz segítségünkre. A következő pontban látni fogjuk az életciklus szerepét is, hiszen kompozíciónál ennek fontos szerepe lesz. Az aggregációt az UML diagramban kitöltetlen rombusszal jelöljük, mint azt az alábbi ábrán is megfigyelhetjük. A kapcsolat azon oldalán tüntetjük fel a rombuszt, amely osztály objektumai a tartalmazó szerepét testesítik meg.

62. ábra: Aggregációs kapcsolat két osztály között

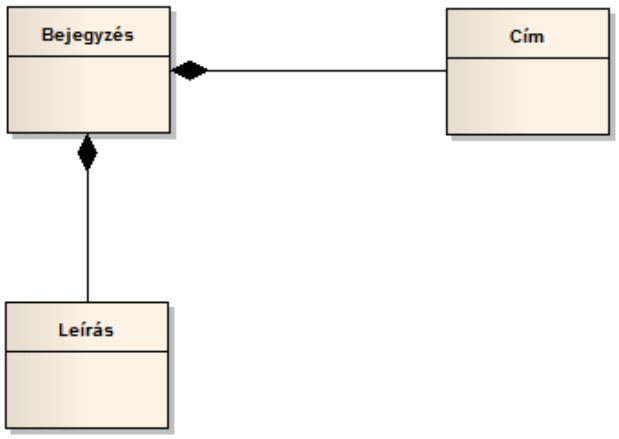
A multiplicitás jelölése nem meglepő a fenti formában: az igények nevű tároló természetes módon számos igényt tartalmazhat.

Kompozíció

Az aggregációnál erősebb kapcsolatot fejez ki két osztály között a kompozíció. Mint azt fent láttuk alapvetően két kérdést kell megvizsgálnunk:

- Fennáll-e a rész-egész viszony két osztály között, és ha igen, ez milyen mértékű.
- Valamint érdemes az életciklus tekintetében is megállapításokat tennünk.

Ha a rész-egész viszony fennáll két osztály között és azt tapasztaljuk, hogy ez fizikai tartalmazást is jelent (pl. egy autónak szervesen része a motor), valamint azt látjuk, hogy a két objektum életciklusa megegyezik, azaz azonos időben jönnek létre az objektumok (nem létezhet autó motor nélkül), akkor kompozíciós kapcsolat bevezetésére lehet szükség. A közösségi oldal bejegyzése kapcsán könnyen találhatunk példát kompozícióra. Tegyük fel, hogy egy bejegyzésnek minden esetben van címe, és ezt a címet követi a tényleges leírás. Ekkor három osztály együttesen fejezheti ki ezt a kapcsolatot: maga a bejegyzés, a címsor, valamint a tényleges törzsszöveg, ahogy azt alább is látjuk:



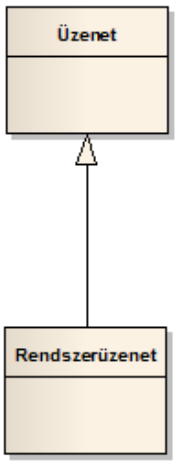
63. ábra: Kompozíciós kapcsolat

Ahogy az a fenti ábrán is látszik, kitöltött rombusz jelöli a kompozíciós kapcsolatot és az a tartalmazó osztály oldalán kap helyet. Ez esetben a multiplicitások minden kapcsolat esetén 1-1 értékűek, azaz egy bejegyzés pontosan egy címet tartalmaz, és pontosan egy leírást.

Öröklődés

Az objektum-orientált filozófia egyik alapeleme az öröklődés, nevezetesen az, hogy a gyermek osztály örökli a szülő osztály adattagjait valamint metódusait. Két osztály között a legerősebb kapcsolatként definiálhatjuk, hiszen közös adattagok és metódusok is jelen vannak. Ahogy arról már szóltunk, a láthatósági opciók közül a védett (protected) jelzőnek van fontos szerepe öröklődés esetén, hiszen ekkor teljesül a megállapítás, miszerint a gyermek osztályban elérhetők az ős adattagjai és metódusai. Az öröklődés során általában két irányt vehetünk figyelembe: haladhatunk az általános osztálytól a

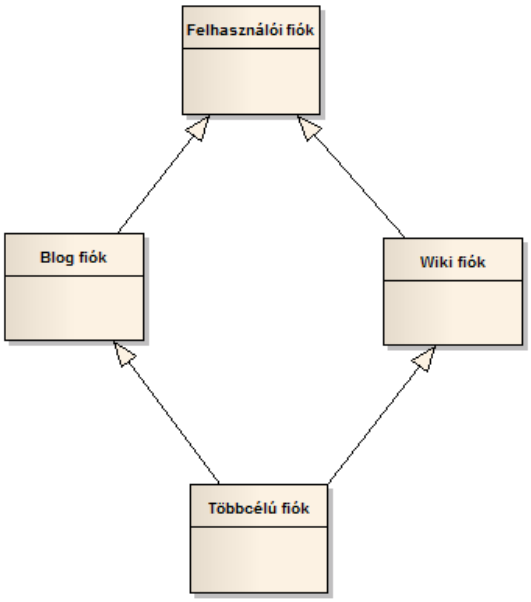
speciális felé, de tekinthetjük ezt a kapcsolatot inverz módon is, mikor általánosításról beszélünk. Ekkor egy/több speciális osztály közös, általános ősét határozzuk meg. Egy egyszerű példával élve a közösségi oldalunkon kétféle üzenetet küldönböztethetünk meg: léteznek felhasználói üzenetek, amelyeket a rendszer használói küldhetnek egymásnak, de létezhetnek speciális rendszerüzenetek is, amelyeket az admin indított útjára, és amelyekben foglaltakat követnünk kell, hiszen ez lehet a rendszer további használatának előfeltétele. Az öröklődés relációt az UML diagramban kis háromszöggel jelöljük, mint ahogy azt lent láthatjuk:



64. ábra: Származtatás

Az öröklődés során érdemes különös gondot fordítanunk néhány speciális esetre:

- Megengedjük-e, hogy egy leszármazott osztálynak több őse legyen?
- Megengedjük-e az un. gyémántöröklődés esetét, amikor egy gyermek osztály valamely felsőbb szintű ősének több adattagját örökölheti különböző öröklődési útvonalon keresztül? A következő diagramot megnézve látható, hogy a többcélú fiók a felhasználói fióktól, mint őstől, két útvonalon is örökli az adattagokat akkor, ha ezt nem kezeljük.

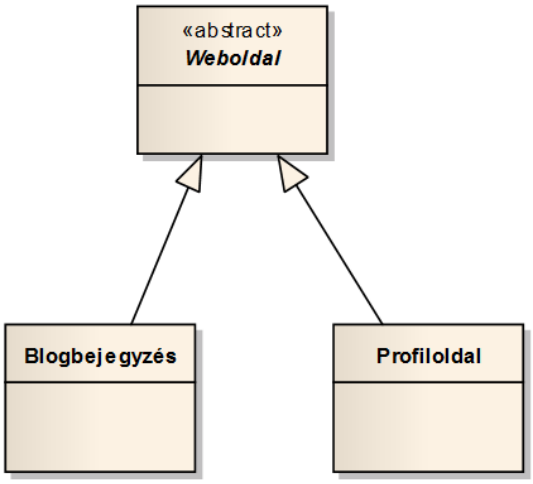


65. ábra: Gyémántöröklődés

A fenti kérdések megválaszolásához előzetes tervezési döntés szükséges, amely többek között az implementálandó rendszer tulajdonságai, valamint az alkalmazott programozási nyelv alapján kerülhet rögzítésre.

Absztrakt osztályok

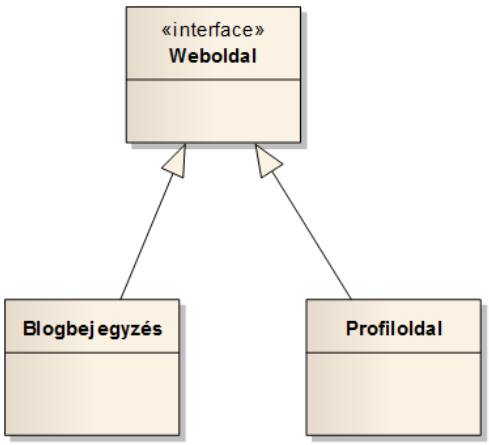
Gondos tervezés eredményeképpen olyan osztályhierarchia jöhet létre, amelyben olyan osztályok is helyet kapnak, amelyekből nem hozható létre objektumpéldány. Fogalmi szinten értelmezhető az adott osztály, azonban szerepe alapján ősként viselkedhet. Gondoljunk például arra, hogy egy blogbejegyzés és egy profiloldal alapja is egy-egy weblap. Éppen ezért jogos tervezési döntésnek tűnik létrehozni egy weboldal nevű osztályt, amelyből mint ősből származtatható lesz a blogbejegyzés illetve a profiloldal. A weboldal mint fogalom tehát megjelenik az osztálydiagramon, de konkrét előfordulása nem értelmezhető, mert általános. Számos előny sorakoztatható azonban fel az absztrakt osztályok alkalmazása mellett. Egyik ilyen előny a polimorfizmus, azaz többalakúság, amely lehetővé teszi számunkra, hogy egymással szülő-gyermek kapcslatban levő osztálybeli objektumokat az ős típus alapján kezeljünk. Ennek számos pozitív hozadéka van, mint ahogy azt objektum-orientált programozásból tudjuk. Az osztálydiagramon több módja van annak, hogy jelöljük egy osztály absztrakt tulajdonságát. Legygyakrabban kiírjuk az osztály neve mellé az **abstract** kulcsszót, de az is gyakran alkalmazott módszer, hogy dőlt betűvel írjuk annak nevét. Az alkalmazott programozási nyelvtől függően aztán a megadott módon kell ezt implementálnunk.



66. ábra: Öröklötetés absztrakt osztályból

Interfészek

Röviden szólunk az interfészek által nyújtott lehetőségekről is. Interfész alatt olyan speciális absztrakt osztályt értünk, amely csak metódusokkal rendelkezik, adattagokkal nem. Az interfészek alkalmazása számos ponton lehet előnyös, hiszen elkerüli például a gyémánt öröklés által okozott problémát. Jelölése UML diagramon az **interface** kulcsszóval lehetséges.



67. ábra: Interfész osztály

7. A mintarendszer osztálydiagramjának elkészítése

Jelen fejezetben a mintafeladat osztálydiagramnájak készítéséhez adunk segítséget. A rendszer számos funkciót tartalmaz, melyeket a korábbi fejezetekben már ismertettünk. Egy rendszer osztálydiagramjának elkészítésekor érdemes elsőként részletesen áttanulmányoznunk az interjú eredményeképpen letisztult feladatmegfogalmazást, hiszen abban elhangzott főnevek és fogalmak megjelenhetnek osztályként a tervezendő alkalmazásunkban. Így jelen feladatnál a következő kulcsszavak fontos szereppel bírnak majd:

- menetrend
- üzenet
- útvonal
- szerelés
- saját menetrend (buszsofőré)
- busz
- diszpécser
- sofőr
- titkár
- szerelő
- szervízkönyv
- szervízkönyv-bejegyzés
- városok

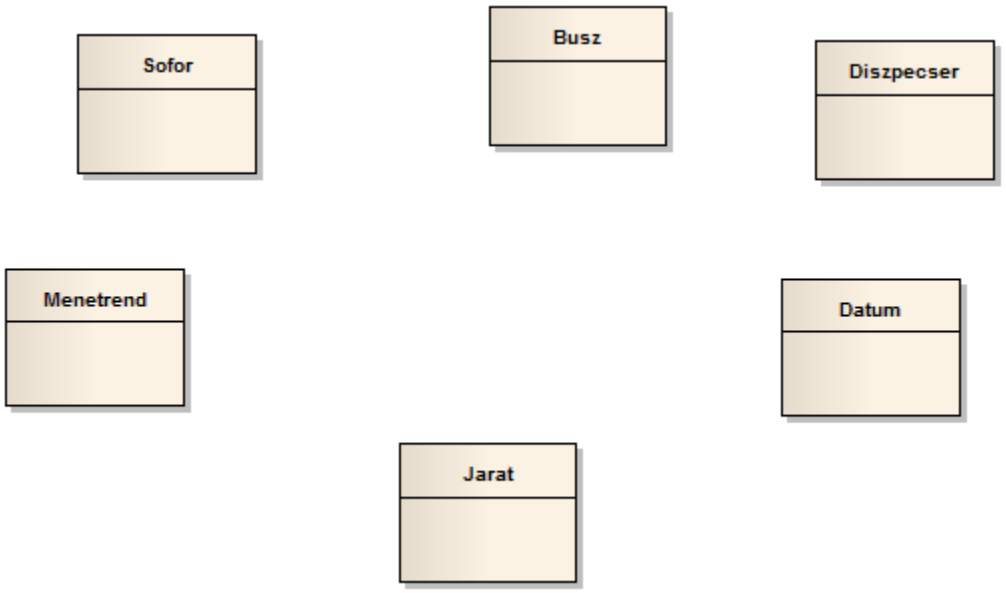
Egy ilyen gyorsan összeállítható felsorolás megadja az elkészítendő diagram alapját és egyúttal abban is nagy segítség, hogy a felderített osztályok közötti kapcsolatokat felmérjük. Mindezek előtt azonban szükséges áttekintenünk a listát és homonímákat, szinonímákat keresnünk, hiszen azokat bizonyosan javítanunk kell.

Következő lépésként az osztályokat jellemző adattagok felderítése lehet feladatunk, majd a metódusok összeállítására is gondot kell fordítanunk. Így tehát az előző fejezetben ismertetett osztályokat bemutató absztakció minden szintje megjelenhet a tervezés során.

A fenti lépések bemutatására a tervezendő szoftver egy központi funkcióját választottuk ki, nevezetesen tudja a diszpécser felvinni, vagy módosítani egy adott sofőrhöz tartozó menetrendet és azt a sofőr tudja is megtekinteni. E forgatókönyv leírásához a következő osztályokra minimálisan szükségünk lesz:

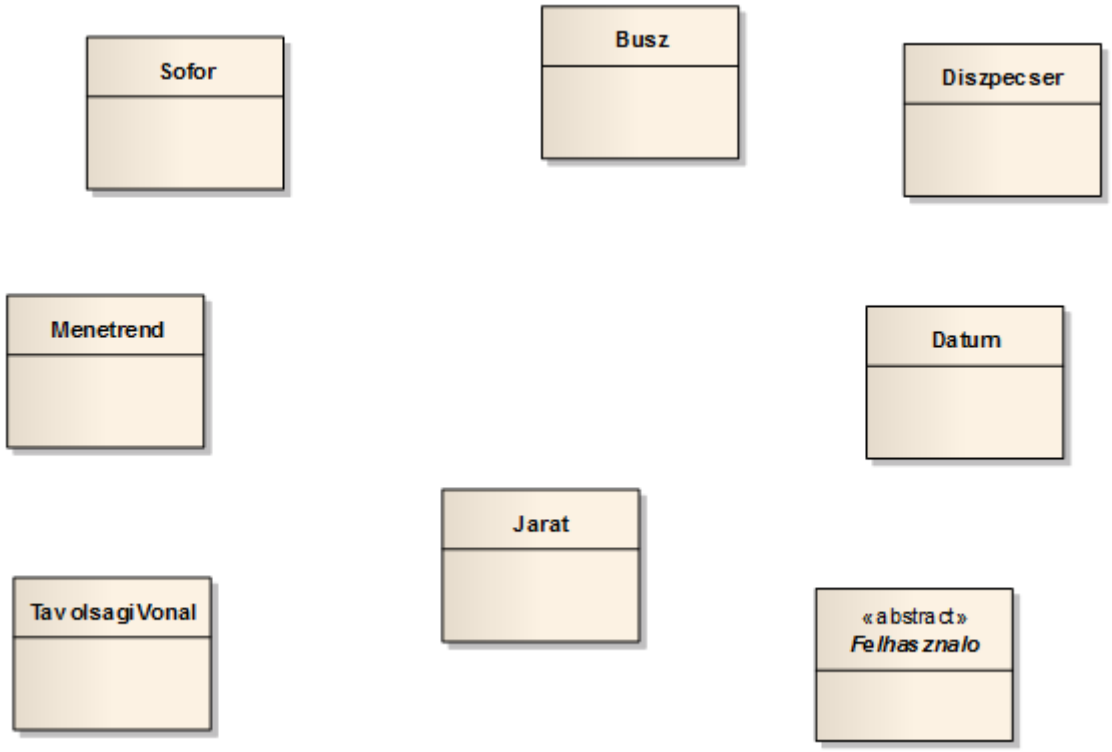
- sofőr(ök),
- busz(ok),

- járat(ok),
- menetrend,
- diszpécser,
- dátum és időpont.



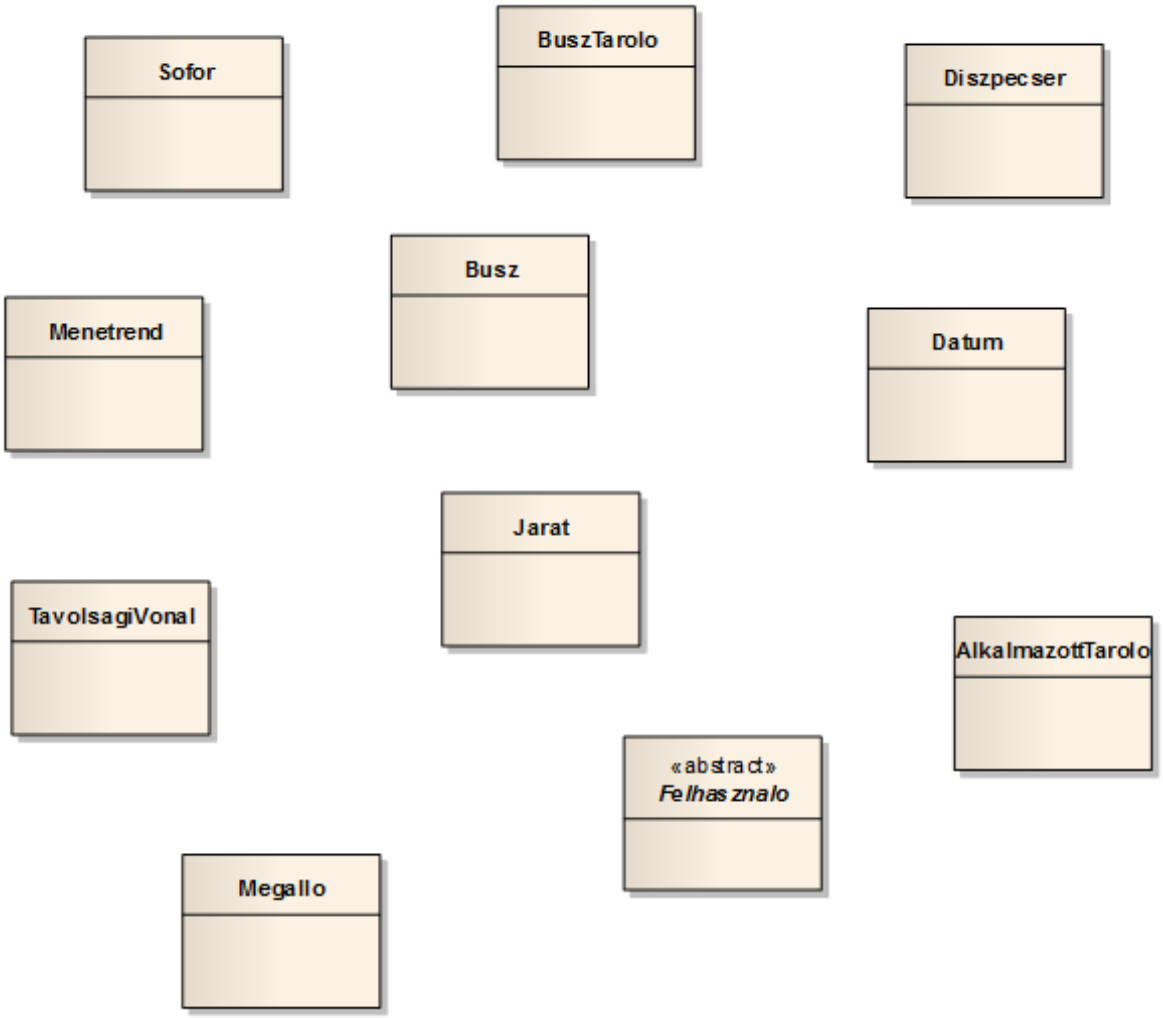
68. ábra: Az első lépésben azonosított osztályok

Mint azt fent láthatjuk szükség lesz a sofőröket, buszokat járatokat leíró osztályokra, és a menetrendet is tárolnunk kell. Ha jobban megvizsgáljuk a problémát az is szembetűnik, hogy a járat osztály további felbontást igényel. Vezessük be a TavolságiVonal osztályt, amelynek feladata az lesz, hogy tárolja azon városokat/megállókat, ahol az adott járatok megállnak. Így például egy távolsági vonalként tekinthetjük a Veszprém, Várpalota, Székesfehérvár, Budapest városokat érintő járatokat, amelyek más helyen nem állnak meg. Azt is vegyük észre, hogy a diszpécser és a sofőr a vállalat egy-egy alkalmazotti köréhez tartozó fogalmak, így célszerűnek tűnik felvenni egy őket jellemző ős osztályt, amelyet nevezzünk Felhasználó osztálynak. E felhasználó osztály legyen absztrakt, hiszen az csak egy általános megnevezése a sofőrnek, valamint a diszpécsernek. Így modellünk tovább alakult:



69. ábra: A finomított osztálylista

Ne feledkezzünk meg arról sem, hogy nem csupán 1-1 objektumot kell tárolnunk, hanem több busz, több felhasználó fogja rendszerünket használni, így olyan tároló osztályokra is szükség van, amelyek az érintett objektumpéldányokat tudják eltárolni. Ezért modellünket következő lépésben kiegészítjük a buszokat és a felhasználókat tárolni képes konténerekkel, valamint egy Megálló nevű osztállyal is, amely azon városokat rögzíti, ahol a buszok megállnak (ez természetesen nem csak város lehet, hanem minden olyan helység, amely a menetrendben szerepel).

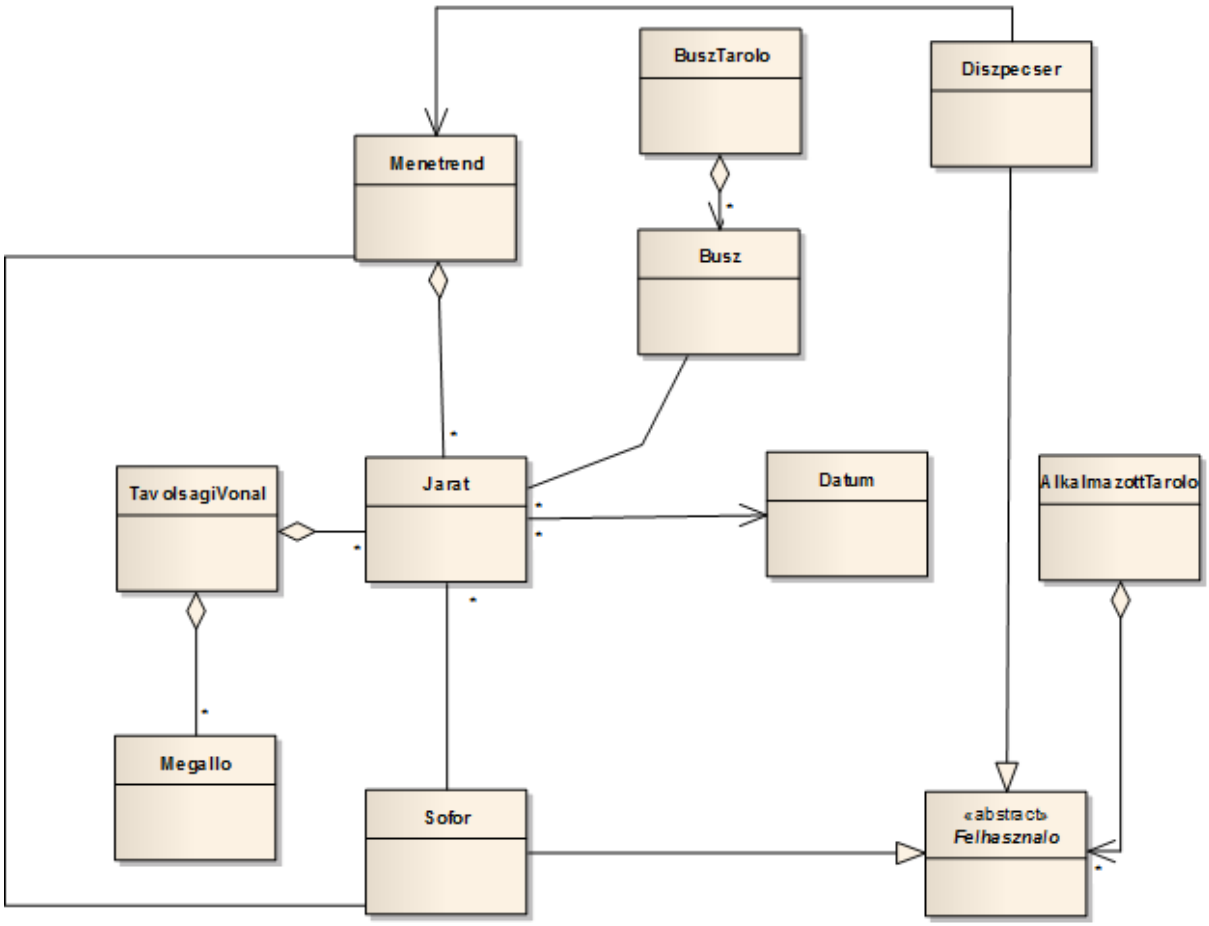


70. ábra: Tárolóosztályokkal kiegészített osztálylista

Következő nagyobb lépésként gondoljuk végig, hogy a megadott osztályok között milyen kapcsolatok definiálhatóak:

- az alkalmazott tároló osztály felhasználókat fog majd tárolni tetszőleges számban;
- egy felhasználó lehet sofőr, vagy diszpécser;
- a buszokat a busztároló osztály fogja eltárolni;
- egy távolsági vonal megállókat foglal magában;
- a menetrend járatok halmazát jelenti;
- egy járatot egyértelműen azonosítják a következők:
  - mely távolsági vonalon közelekedik
  - ki, mely buszsofőr bonyolítja le az adott járatot
  - mely busszal kerül teljesítésre az adott járat
  - mikor indul és érkezik a járat.

A fentieknek megfelelően alkalmazva az UML jelöléseket a következő struktúrájú osztálydiagram rajzolható fel:



71. ábra: A mintapéldához tartozó már kapcsolatokat is tartalmazó osztálydiagram

Következő lépésként gyűjtsük össze az osztályokhoz tartozó adattagokat.

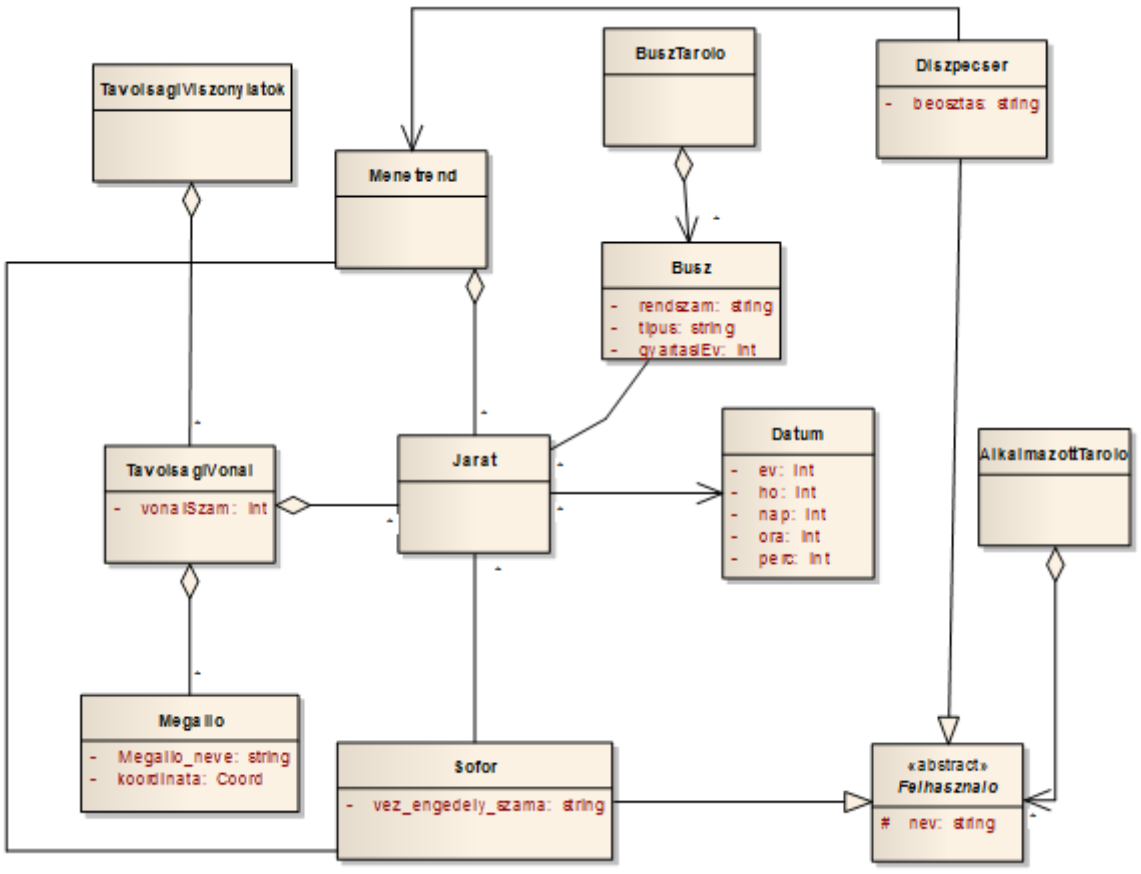
Az AlkalmazottTarolo osztály feladata, ahogy a neve is mutatja felhasználók tárolására korlátozódik. A Felhasználó osztály absztrakt osztály, így itt kihasználjuk a polimorfizmus adta lehetőségeket, így ős típus alapján tároljuk az objektumokat.

Egy felhasználó csupán egy névvel rendelkezik most, de természetesen számos más személyes adat megadható lenne itt. Ezen ősosztályból két gyerekosztály került származtatásra: minden sofőr rendelkezik vezető engedéllyel, így annak azonosítóját tároljuk, illetve diszpécserek esetén fontos eltárolnunk a beosztás megnevezését. Sofőrök, mint felhasználók között nem teszünk különbséget beosztás tekintetében. A diagram központi eleme a járat osztály. Mi határoz meg egy járatot?

- az a vonal, amelyen az közlekedik;

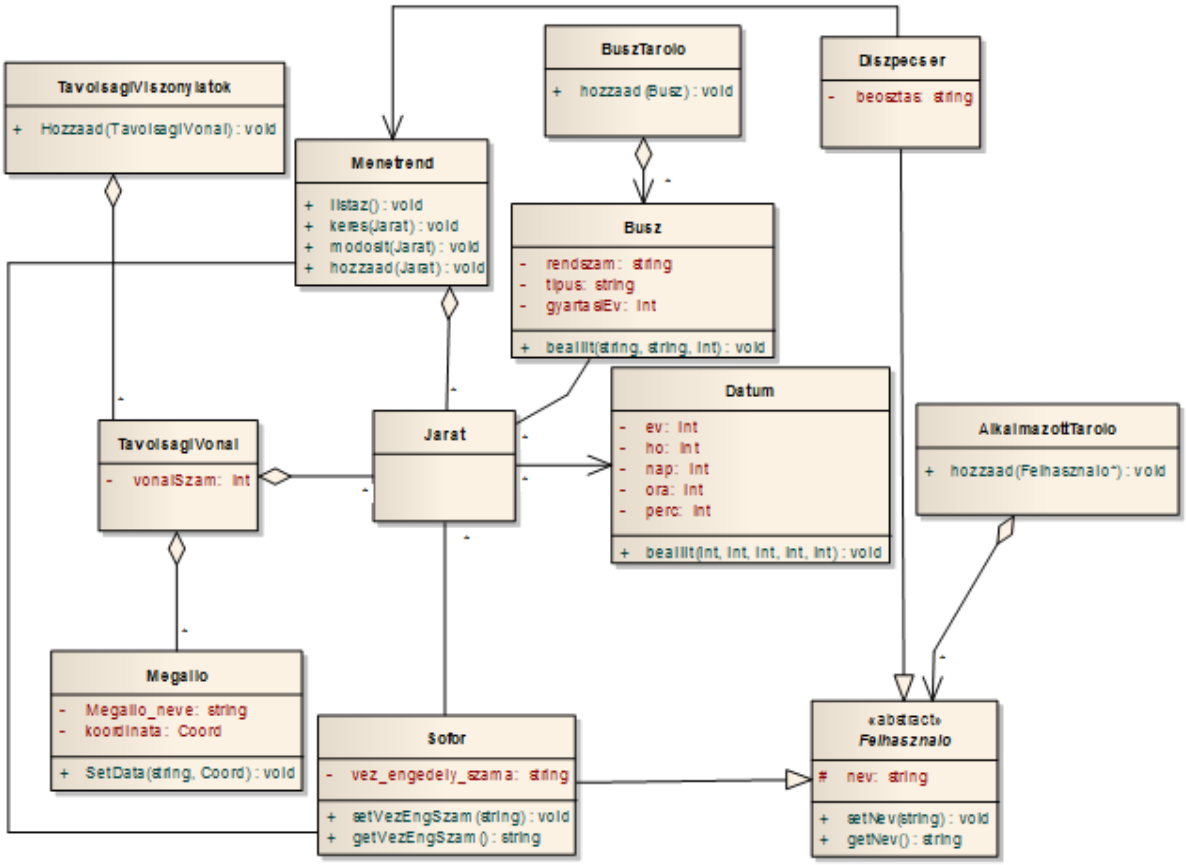
- az a sofőr, amelyik az adott járatot vezeti;
- az a busz, amellyel a járat teljesítésre kerül, valamint
- egy dátum évvel, hónappal, nappal, órával és perccel, amikor az elindul.

Természetesen egy ilyen konkrét járat eleme kell, hogy legyen a menetrendnek, valamint a távolsági vonal osztály is megállók sorozatát tárolja. Észrevehető, hogy jelen leképezés egy távolsági vonalat kezel, éppen ezért célszerű kiegészítenünk a modellt úgy, hogy több vonal tárolása is megoldható legyen. Egy távolsági vonalat annak száma és a megállói jellemeznek, míg egy buszt annak rendszáma, típusa és gyártási éve ad meg. A menetrend, a busztároló, további tároló osztályok. Adattagokat külön nem adunk itt meg, hiszen a kapcsolatok révén a szükséges referenciák/mutatók az osztály részei.



72. ábra: Adattagokkal kiegészített osztálydiagram

Az osztálydiagram tervezésének következő állomásaként a metódusok meghatározását kell elvégeznünk.



73. ábra: Metódusokat is tartalmazó osztálydiagram

Fontos hangsúlyoznunk, hogy számos metódus let volna definiálható a fenti ábrán. Célunk most az volt, hogy a diszpécsernek jogot és lehetőséget adjunk arra, hogy a menetrendet listázza, illetve módosításokat hajtson végre. Ezeket meg tudja tenni, hiszen a Menetrend oszály megfelelő metódusai rendelkezésre állnak, ezekkel a kívánt műveletek megvalósíthatóak.

Számos más alternatív osztálydiagram is megadható a feladat helyes megoldására. Tervezési minták alkalmazásával, interfészek bevezetésével még inkább objektum-orientálttá tehető a diagram.

8. Interakciók a rendszerben: szekvenciadiagramok készítésének alapjai

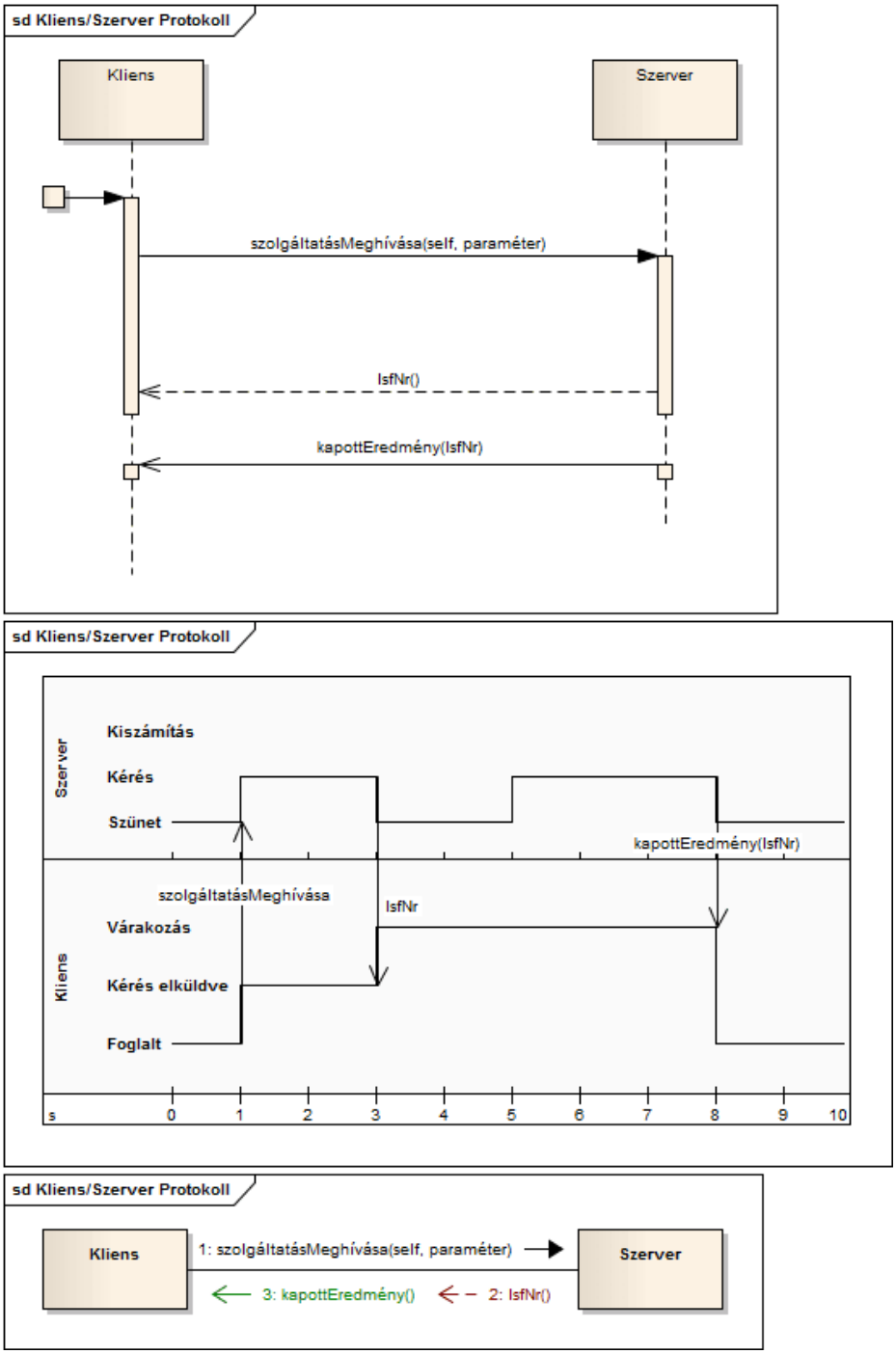
Az interakciós diagramoknak három fő típusát találjuk meg az UML-ben. Ezek a szekvencia-, az idő- és a kommunikációs diagramok. Mindegyik az üzenetek küldését írja le a rendszerben, de más fókuszponttal. A szekvenciadiagramok esetében az üzenetek cseréjére, az idődiagramoknál az időbeli lefutásra, a kommunikációs diagramnál pedig a struktúrára helyezi a hangsúlyt. A negyedik diagramtípus, amit meg kell említeni az interakciós diagramoknál, az az interakciós áttekintés. Tulajdonképpen tevékenységdiagramnak tekinthető, de a szekvenciadiagramok áttekintését segíti azok összekapcsolásával.

Az interakciós diagramok lényege tehát a kölcsönhatások ábrázolása. Egy interakciós diagram életvonalakból áll. Ezek az életvonalak az esemény előfordulások következményeként jelennek meg, melyek nem mások, mint az üzenetek a résztvevők között. Esemény előfordulásnak számít egy osztály példányosítása vagy egy objektum önmegsemmisítése is. Az interakciókat lehetséges operátorokkal módosítani, illetve komplex interakcióba szervezni.

Osztályok közötti interakciók: szekvenciadiagramok, időd diagramok, kommunikációs diagramok

Az osztályok és objektumok közötti interakciók a kölcsönös metódushívásokat jelentik az osztályok és objektumok között. Nagyon gyakran használt eszköz a szoftverek életciklusában.

Interakciók bemutatására egy ismert példát nézünk meg, a kliens/szerver kölcsönhatást. Ezt mind a három már említett interakciós diagramon lehet szemléltetni.



74. ábra: Kliens/szerver kölcsönhatások szekvenciadiagramon, idődiagramon és kommunikációs diagramon

Az interakciós diagramban a partnerek megjelenésének sorrendjét meghatározza az üzenetküldések sorrendje. Az először kapcsolatot kialakító partner lesz balról az első a diagramon. Partnereket tehát meghívásuk sorrendjében kell szerepeltetni. Minden partner esetében szerepel az életvonal, amely fentről lefelé mutatja az idő múlását. Az életvonal szaggatott vonallal is ábrázolható. Ahogy már említésre került, az életvonalak mentén esemény-előfordulásokat találunk, amelyek lehetnek

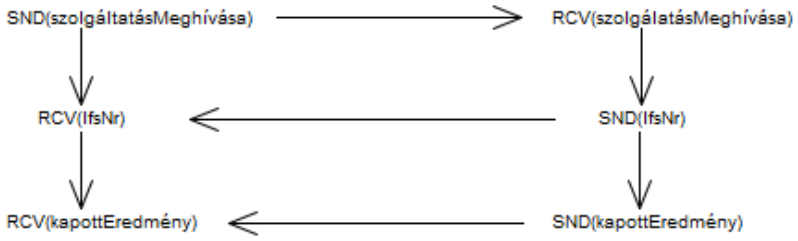
üzenetküldések vagy fogadások. Ezeket az üzenetküldéseket és fogadásokat nyíllal jelöljük, a küldőtől a fogadó felé. Az ábrán különböző üzenettípusokat láthatunk. Az első üzenetben a kliens egy szolgáltatásMeghívása üzenetet küld két paraméterrel a szervernek. Erre válaszul a szaggatott vonallal rajzolt nyílon megjelenik egy visszatérési érték a szervertől a kliens felé. Időben kicsit később a szerver elküldi a kapottEredmény-t a visszatérési értékkel azonosítva a kliensnek. Az üzenettípusok szinkron és aszinkron üzenetek lehetnek a partnerek között. Bár a nyilak közötti távolság nem azonos, ez semmilyen jelentőséggel nem bír az üzenetek közötti időbeli távolság szempontjából. Ezen a típusú interakciós diagramon nem lehet konkrét időbeli lefutást mutatni az egyes üzenetekre vagy az üzenetek közötti távolságra. A szekvenciadiagramok akkor használhatóak jól, ha az interakció kevés partner között zajlik, és ők sok üzenetet cserélnek. Az interakciós minta lehet komplex, kevés partner esetén ennek megjelenítése és értelmezése nem jelent gondot a diagramon. A partnerek állapotait és a már említett időtényezőt csak nehezen lehet rajta ábrázolni.

A második lehetőség az üzenetváltások bemutatására illetve modellezésére a kommunikációs diagram. Az ábrán az előzőleg bemutatott üzenetváltás látható. A kliens és a szerver közötti kapcsolatot egy összekötő mutatja. A köztük lévő üzenetek nyilakkal és sorszámokkal valamint a meghívott metódusokkal vannak jelölve. A kommunikációs diagramtípus akkor használható jól, ha több partner vesz részt az interakcióban, komplex hálózatot jól lehet vele ábrázolni. Az üzenetek számának növekedése azonban rontja a diagram átláthatóságát.

Az interakciós diagramok harmadik típusa az idődiagram, amely az ábrán látható módon ugyanazt a kliens/szerver interakciót mutatja be. A függőleges tengelyen a résztvevők láthatóak, állapotaikkal megjelenítve. Az állapotok megjelenítése nem előírása a diagramnak. A vízszintes tengely az időtengelyt ábrázolja, amelyen az időbeli távolságok egyenesen arányosak a vízszintesen mért távolságokkal. Használható szimbolikus időskála is: ezen csak egyes időpontokat nevezünk meg explicit módon. Az interakciós partnerek életvonalai itt vízszintesen balról jobbra futnak. Függőlegesen láthatjuk az egyes üzenetküldéseket és az üzenetküldésekre bekövetkező állapotváltozásokat az interakciós partnereknél. Az üzentek küldése itt is nyilakkal van jelölve, mellette az üzenet típusával. Ezzel a diagramtípussal a pontos időbeni koordinációt, a valós időben zajló folyamatokat ábrázolhatjuk jól. A partnerek számának növekedésével és az interakciók komplexé válásával az áttekinthetősége csökken.

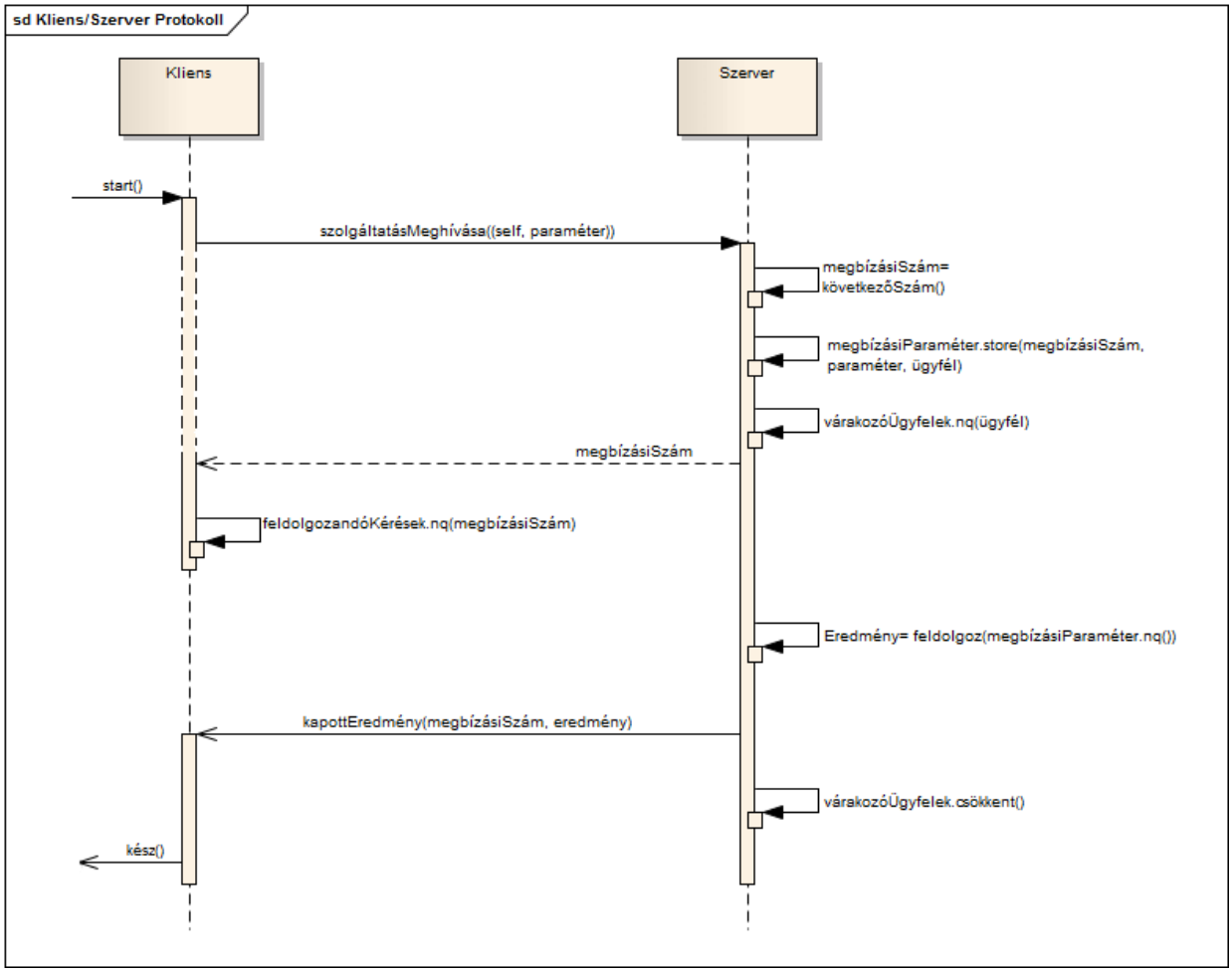
Interakciók jellemzése

Az interakció interakciós események egymásutániságának folyamata. Az események egymásutániságának sorrendjét két szabály határozza meg: az egyik, hogy az üzenet küldése mindig megelőzi az üzenet fogadását, a másik pedig, hogy az esemény előfordulások sorrendjét az életvonalon való megjelenésük sorrendje határozza meg. Ennek a két szabálynak a leírása alapján az alábbi ábra mutatja az előző ábrákon látott kliens/ szerver kapcsolat eseményeinek egymásutániságát. Természetesen ebből nem tudunk meg semmit a pontos időbeni távolságokról.



75. ábra: Az esemény-előfordulások egymásutániságának összefüggései

Az üzenettípusok a partnerek között többfélék lehetnek. Az életvonalak mentén nem csak az üzenetek küldését és fogadását láthatjuk, hanem úgynevezett aktiválási sávokkal azt is megmutathatjuk, hogy eközben a partner aktív vagy passzív állapotban van e működését tekintve. A példában az interakció kezdetekor a szerver és a kliens is inaktív állapotban van. Első lépésben a start üzenet folytán a kliens aktiválásra kerül, és a szolgálatatásMeghívása üzenettel aktiválja a szervert. Innentől az ábrán szaggatott sávval jelölve látszik, hogy a kliens inaktív állapotban lesz. A szerver a szolgáltatásMeghívása üzenet fogadása után egy j megbízási számot ad a megbízásnak, majd pufferbe (megbízásParaméter.store) helyezi, ezzel együtt a kliensre vonatkozó referenciát pedig a várakozóÜgyfelek sorba teszi be. A megbízási szám visszakerül a klienshez, akinek az aktiválása innentől folytatódik, és a megbízási számot a feldolgozandóKérések sorba veszi fel. Egy bizonyos idő után – amit a hullámos vonalak jelölnek – a szerver elkezdi a kérés feldolgozását. A szerver fő vezérlési vonala megszakad és aktiválási sáv jön létre az aktiválási sávon belül. A feldolgozás után az eredményt a szerver a megbízási számmal azonosítva elküldi a kliensnek.



76. ábra: Kliens/szerver protokoll üzenetküldései

Az üzenettípusok az interakciós partnerek között lehetnek szinkron vagy aszinkron üzenetek. A szinkron üzenteknél csak az egyik partner van aktivált állapotban, az aszinkron üzenteknél mindkettő aktivált lehet. A következő típusaikról beszélhetünk:

- Kérés: szinkron üzenet, amely során a küldő aktivitása megszűnik, és a fogadó kerül aktív állapotba. Ábrázolása: küldőtől fogadó irányába telített hegyű nyíl.
- Válasz: szinkron üzenet, amely egy megelőző kérésre vonatkoztatható. Elküldésével a küldő aktivált állapota megszűnik, és a fogadó addigi deaktivált állapotból újra aktivált állapotba kerül. Jelölése: szaggatott nyitott hegyű nyíl a küldőtől a fogadó irányába.
- Szignál: aszinkron üzenet, tehát a küldő aktivált állapota változatlan marad. Jelölése: küldőtől fogadó irányába mutató nyitott hegyű nyíl.

Az aktivált állapotba kerülést biztosító szinkron üzenetet nem csak deaktivált állapotban kaphatja meg a fogadó. Ha a fogadó aktív, és így érkezik hozzá egy szinkron kérés, akkor egy újabb aktiválási sáv kerül feltüntetésre az aktiválási vonalán. Hasonló gondolatmenetet követve, nem kerül mindenképpen deaktivált állapotba a szinkron választ küldő partner, de az aktiválási sáv mélysége eggyel csökken,

tehát ha három mélységű aktiválási sávval rendelkező állapotban volt a küldés pillanatában, akkor a sávok mélysége eggyel csökken, és kettő mélységű lesz az üzenet elküldése után.

Az üzentek nyelvtana a következő:

Üzenet ::= Szignál|Kérés|Válasz

Szignál ::= Szignálnév [Argumentumok]

Kérés ::= Műveletnév [Argumentumok]

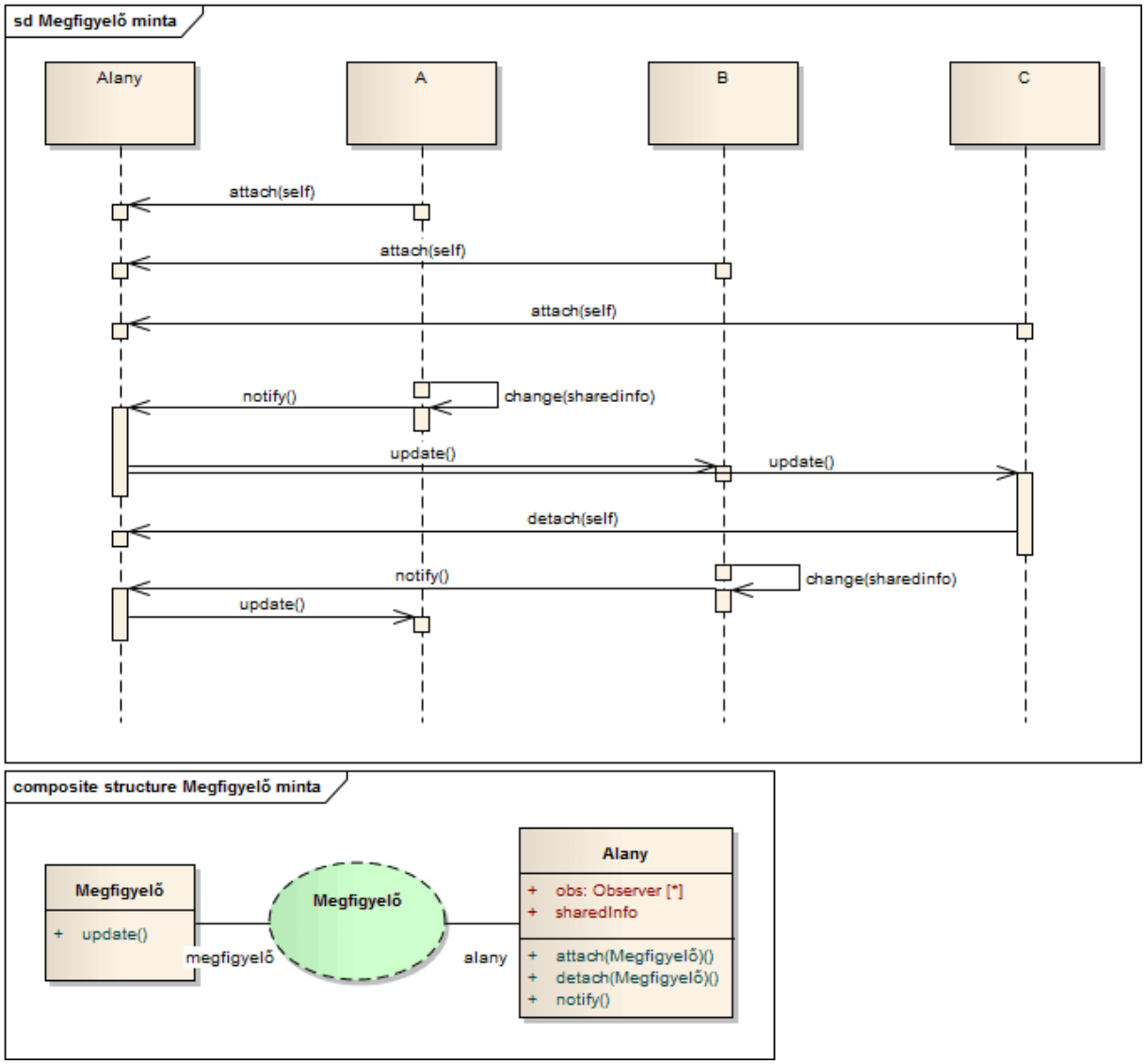
Válasz ::= [Kérés:] VisszatérésiÉrték

Argumentumok ::= (Argumentum (,Argumentum)*)

Argumentum ::= [Irány] Paraméter: Érték|Érték|-

Irány ::= in|out|inout|return

Az interakciókat szívesen használják elemzési és fejlesztési minták dinamikai nézőpontjának vizsgálatára. Az egyik alap együttműködés, amelynek az interakciós diagramját most megvizsgáljuk a megfigyelő minta. A megfigyelő mintában egy úgynevezett Alany szerepel és hozzá több Megfigyelő csatlakozik. Az Alany rendelkezik a hozzá kapcsolódó résztvevők aktuális listájával valamint olyan műveletekkel, amelyekkel a megfigyelők képesek fel és lecsatlakozni. Az Alany tulajdonképpen a megfigyelők által kiosztott információkat osztja meg a megfigyelők között.



77. ábra: A megfigyelő tervezési minta interakciós diagramja és struktúrája[1]

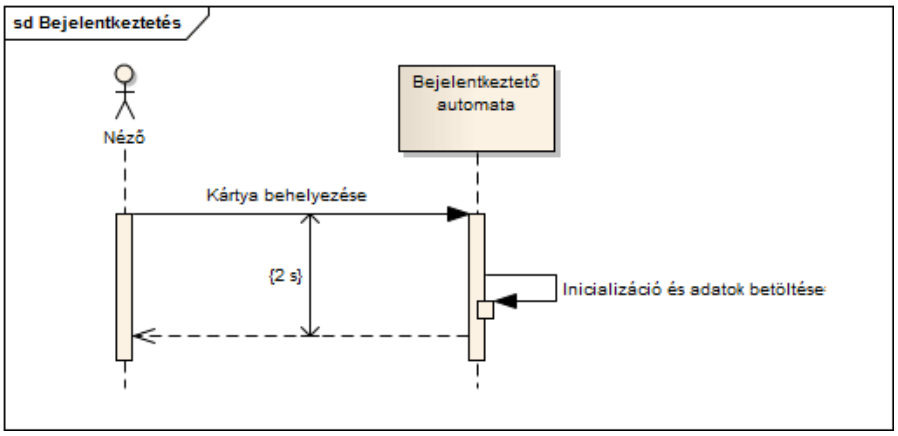
Az ábrán **Error! Reference source not found.**látható módon a megfigyelők az alanynál bejelentkeznek az attach() művelettel, egy önmagukra utaló referenciával. Ha az A megfigyelőnél változik olyan információ, amely a többiekre is vonatkozik, akkor ezt egy notify() üzenettel jelzi az Alanynak aki értesítést (update) küld erről a többi Megfigyelőnek. Ha egy megfigyelő a detach(self) üzenettel lekapcsolja magát a megosztásról, akkor a további frissítésekről már nem kap értesítést.

Kontextusinterakciók

A kontextus-együttműködés a rendszer és a környezete közötti együttműködést mutatja be. A dinamikus viselkedését kontextusinterakciókkal lehet modellezni. A **Error! Reference source not**

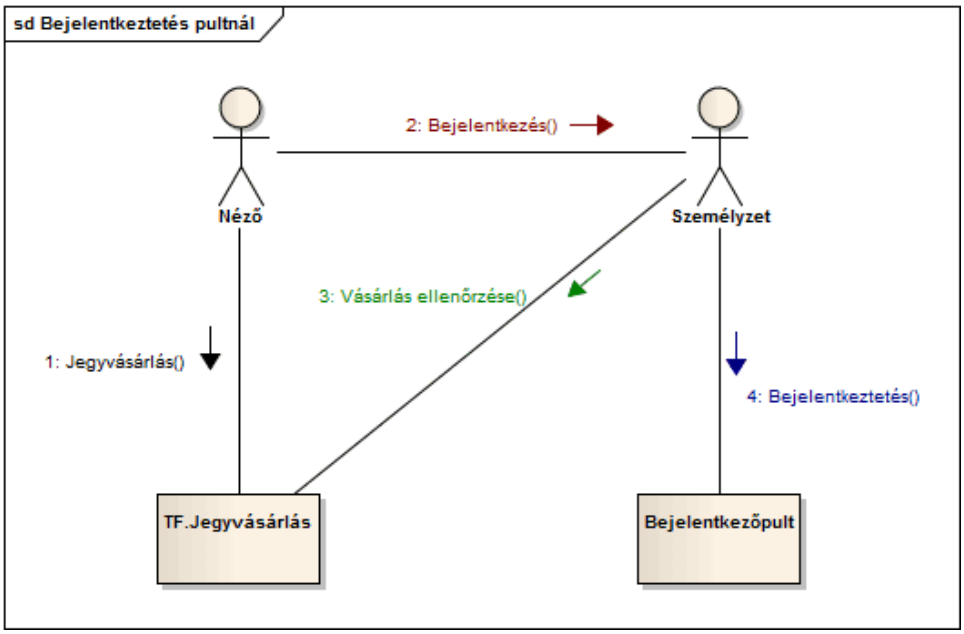
[1] H. Störrle, UML 2 – Unified Modeling Language, Panem Könyvkiadó, 2007.

found. látható, hogy a Néző behelyezi a kártyát a Beléptetőautomatába. Az automata inicializálja magát és rákeres a Néző adataira, majd üdvözli és megmutatja az ülőhelyét a helyszín térképén.



78. ábra: Kontextusinterakció

Egy újabb példa látható az alábbi ábrán:



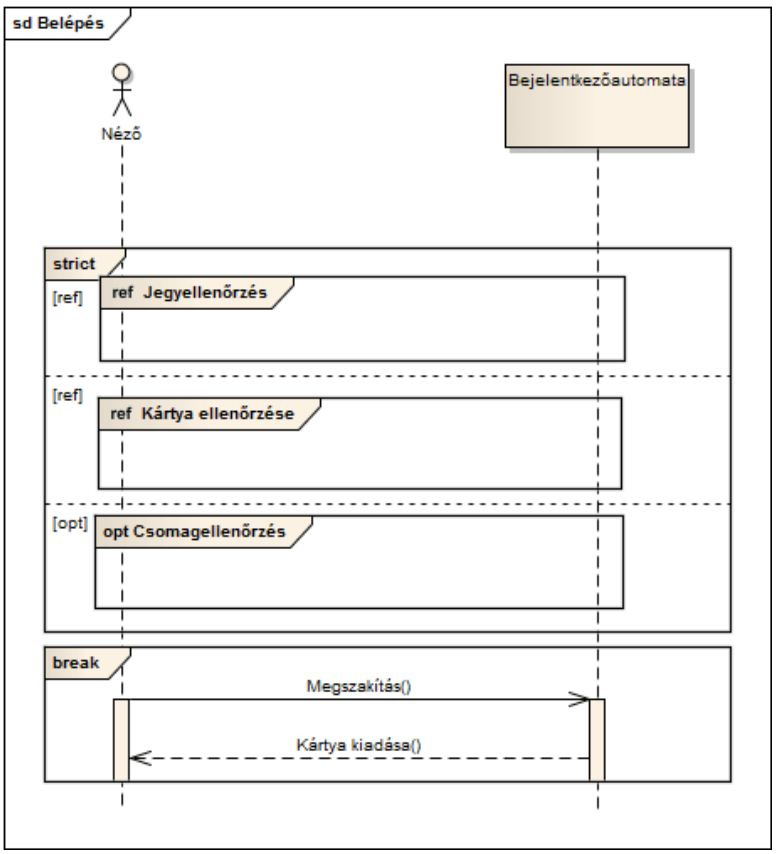
79. ábra: Interakció több partner esetén

Az eset akkor következik be, amikor a kártyás azonosítással valamilyen gond van, és a Személyzetnek kell a Néző segítségére sietnie. Saját képernyőjén kártya számának beírásával vagy a Néző személyes adatai alapján elvégzi a jegyellenőrzést és a Néző azonosítását

A kevesebb, például két partner között lezajló interakcióknál érdemes lehet táblázatos formában leírni az interakciót. A táblázatban a felhasználói műveletek vagy a rendszerrel kontextusban álló partner műveletei és a rendszernek erre adott válaszai kerülnek feltüntetésre.

9. A mintarendszer kiemelt interakciójához szekvenciadiagram készítése

Eddig olyan folyamatokat vizsgáltunk, amelyek egymás utáni lépésekből álltak. Lehetnek azonban olyan esetek a folyamatban, amelyek nem biztos, hogy lefutnak, csak bizonyos feltételek hatására, vagy többször is lefutnak egymás után. Az ilyen esetek kezelésére vezették be a komplex interakciós operátorokat, amelyek segítségével több interakciót kombinálhatunk, így folyamatok egész halmazát írhatjuk le.



80. ábra: Interakciós operátorok

Strict operátor

A belépés folyamata során a rendszer és a felhasználó közötti interakció négy szakaszra bomlik. A Néző először beteszi a jegyet az automatába. Ezután beteszi a hitel- vagy bónuszkártyáját. A következő lépésben a csomagellenőrzésnél, ha van csomagja, akkor a csomagot alhelyezi a csomagmegőrző egy rekeszében, amelyet a bónusz vagy hitelkártyája leolvasásával kérvényez a csomagmegőrző automatájánál. A Néző bármelyik lépésben kiveheti a kártáját az automatákból, ezzel megszakítva a folyamatot. Az ábrán a strict, ref, opt és brk operátorokat látjuk. Az operátorok az interakció felső keretében vannak jelölve. A strict egy komplex interakció, négy operandusa van, amelyek szaggatott

vonallal vannak elválasztva egymástól. Az operandusok maguk ref és opt operátorokkal ellátott interakciók. A strict operátor azt szabja meg, hogy a komplex interakcióban az operandusok szigorú sorrendben követik egymást, és egy szakasz akkor veszi kezdetét, ha az előző teljesen befejeződött.

Ref operátor

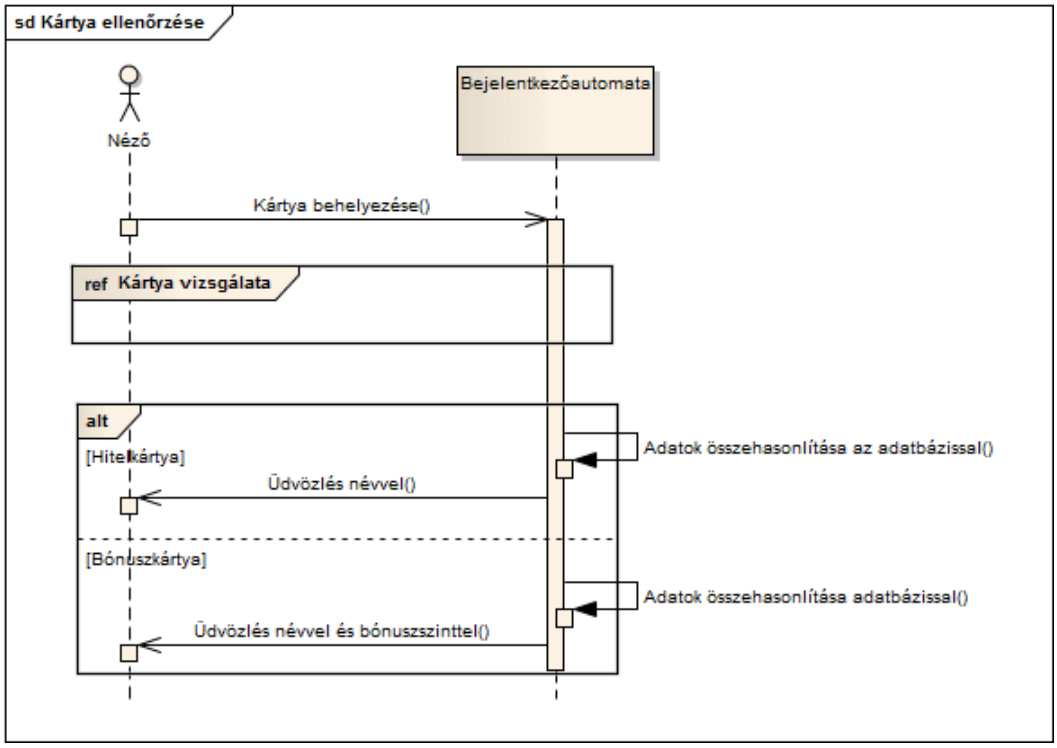
Az operátor más interakciókra való hivatkozást jelöl. Ezek az interakciók más diagramokon lehetnek kifejtve, de tartalmuk akár a strict közvetlen operandusaként is megjelenhetne.

Opt operátor

Az opt segítségével azt tüntetjük fel, hogy egy operandus fellépése feltételes, opcionálisan fellép vagy nem lép fel. Ilyen a folyamatban a csomag elhelyezése megőrzőben, hiszen nem biztos hogy a Nézőnél van csomag amit be kell tennie a csomagmegőrző automata rekeszébe.

Alt operátor

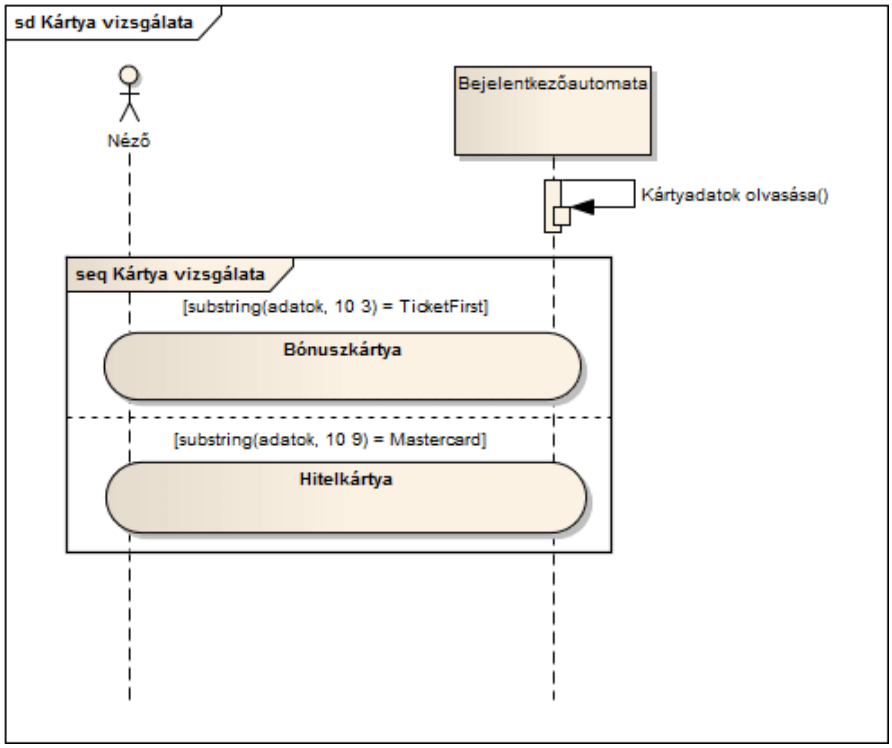
Ha finomítjuk a Néző azonosítása operátort, akkor a különbséget kell tenni a hitelkártyával és a bónuszkártyával történő azonosítás között. Ezt az alt operátorral tehetjük meg. Az alt operátor jelen esetében két operandussal rendelkezik, amelyek alternatív szituációkat jelentenek.



81. ábra: Kártya ellenőrzése szekvenciadiagram operátorokkal

Az alt és az opt operátorok feltételek alkalmazásával irányíthatók. Bonyolult feltételek esetén ugrójelek alkalmazhatók. Ezeket oválisok segítségével ábrázolhatjuk, amelyek a releváns életvonalakon keresztül

jelennek meg. A kártya vizsgálata interakción két ugrójelet, a Bónuszkártyát és a Hitelkártyát azonosíthatunk, amelyek a Néző azonosítása esetén esetválasztás feltételeiként kerülnek elő.



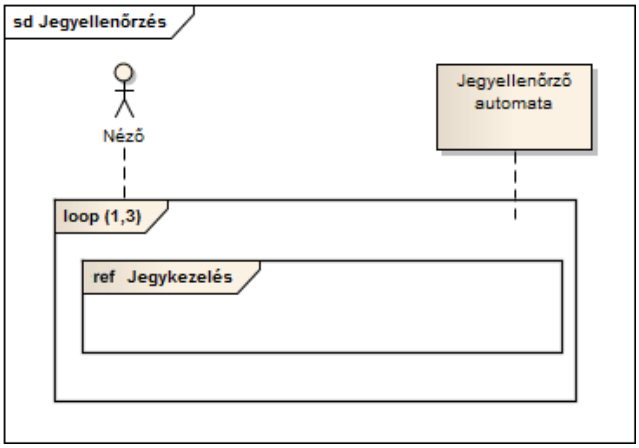
82. ábra: Ugrójelek használata

Brk operátor

Az interakciók megszakítására használatos a brk operátor. Jellemzője, hogy bekövetkezésekor a legközelebbi környezetben lévő interakciót szakítja meg. A példán a brk operátor a Belépés interakciót szakítja meg.

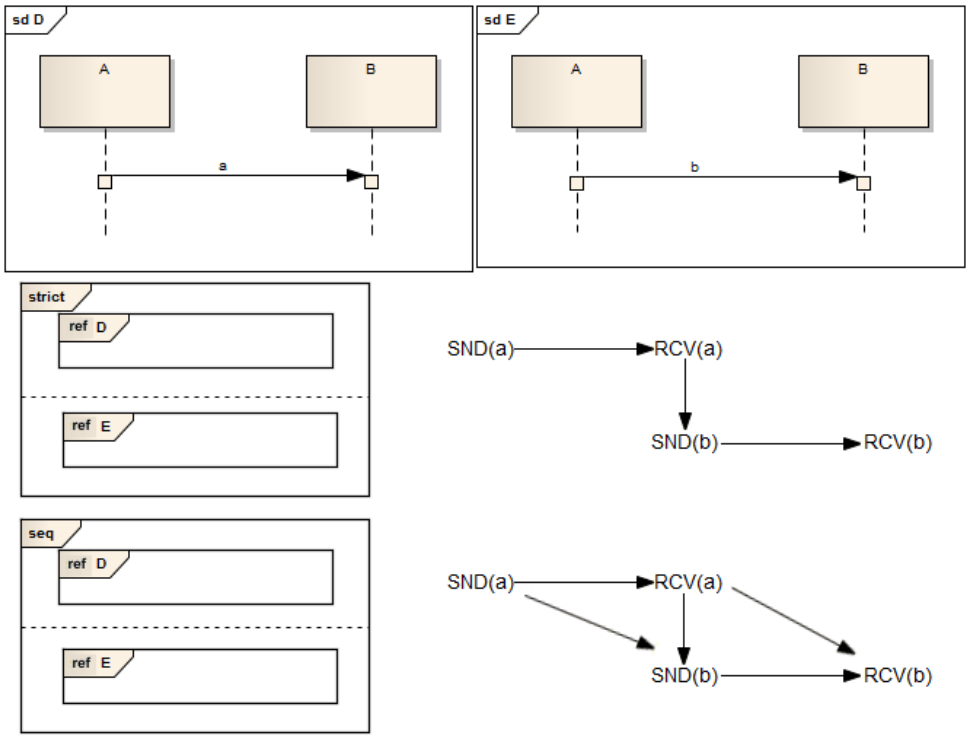
Seq és loop operátorok

A Néző több jegyet is kezeltethet az automatánál. Ennek a megszorításnak a kezelésére alkalmas a loop operátor. A loop operátor az operanduson kívül tartalmazzák a ciklus ismétlődésének alsó és felső korlátját. Felső korlátnak megadható a * érték is, amely tetszőleges számú lefutást takar. Ha nincs megadva az alsó határ, akkor az alapértelmezett érték a 0.



83. ábra: Loop operátor

A seq operátor sorba kapcsol két interakciót. Itt a sorrendiség az egyes életvonalak mentén értendő, nem pedig magán az inerakción véve. Ha több közös életvonal létezik, előfordulhat, hogy a második interakció eseményei következnek be, bár az első interakció még nem fejeződött be.

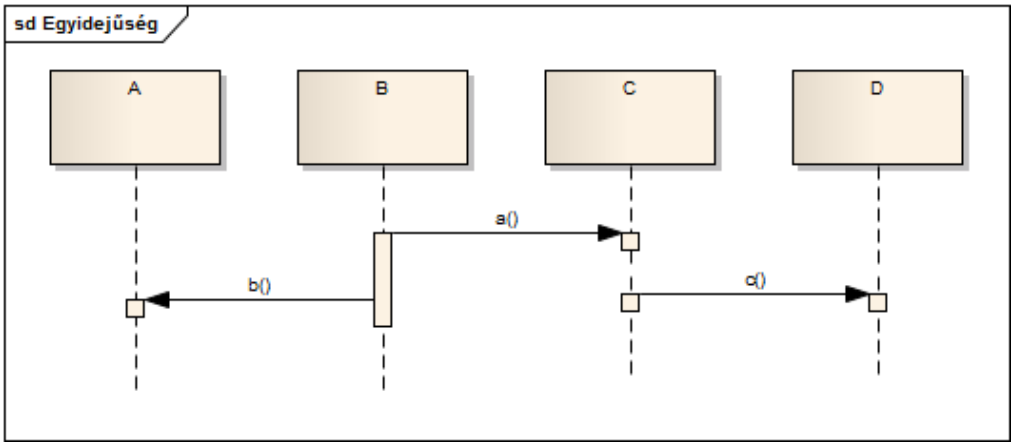


84. ábra: A seq és a strict operációk közötti különbség

A D és a E interakcióban egy a illetve b üzenet küldésére és fogadására kerül sor. A küldési esemény (SND(a)) a fogadási esemény (RCV(a)) előtt zajlik. A strict-el való összekapcsolás miatt a D összes esemény-előfordulása az E első esemény-előfordulása elé kerül. A seq alkalmazásával a D első esemény-előfordulása kerül az E esemény-előfordulása elé.

Par és region operátorok

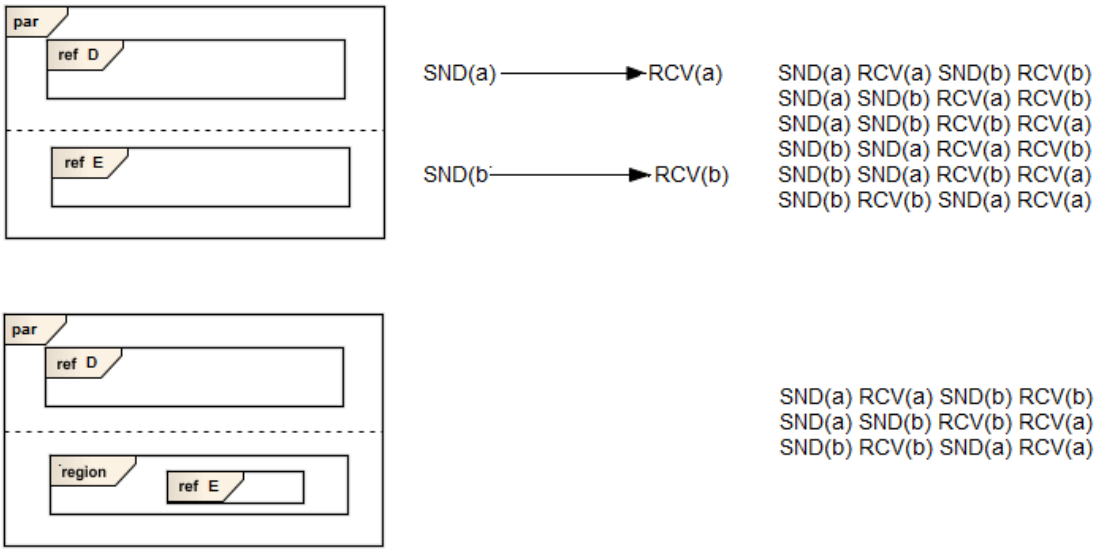
Az egyidejűséget az interakciós diagramról nem lehet egyértelműen leolvasni, mivel az interakciós partnerek köztes időben fellépő állapotairól és egyéb interakcióiról nem esik szó. Így például nem lehet egyértelműen eldönteni az ábrán, hogy elsőnek a b vagy a c üzenet kerül korábban elküldésre.



85. ábra: Esemény előfordulások sorrendje szekvenciadiagramon

Az is lehetséges, hogy először a c fogadása történik meg és aztán kerül a b elküldésre, mivel az életvonal nem időtengely, és a vízszintes sorrendiségből nem következtethetünk az időbeli viselkedésre. Az esemény-előfordulások közötti függőséget lehet specifikálni, amelynek jelölése pontozott vonallal történik, közepén tömör hegyű nyíllal, amely a független esemény előfordulástól a függő előfordulás felé mutat.

Ha a párhuzamos feldolgozást szeretnénk modellezni az interakciók között akkor a par operátor alkalmazható. A par(P, Q) jelentése az, hogy a P és a Q esemény-előfordulásai tetszőlegesen keveredhetnek addig, amíg a P és Q által definiált sorrendiségnek megfelelnek.



86. ábra: A par és a region operátorok hatásai az interakciók lefutására

A region operátorral ezt a halmazt szűkíthetjük, mivel az operátor szerepe az, hogy az általa meghatározott esemény előfordulások nem szakadhatnak félbe.

További interakciós operátorok

A fentieken kívül, számos más operátort használhatunk még az interakciók specifikálására. Az eddig említettek a leggyakrabban használtak közé tartoznak, de tegyünk röviden említést a továbbiakról is.

ignore(P,Z): az üzenetek Z halmazát deklarálja, amelyet a P interakció esetén figyelmen kívül szeretnénk hagyni

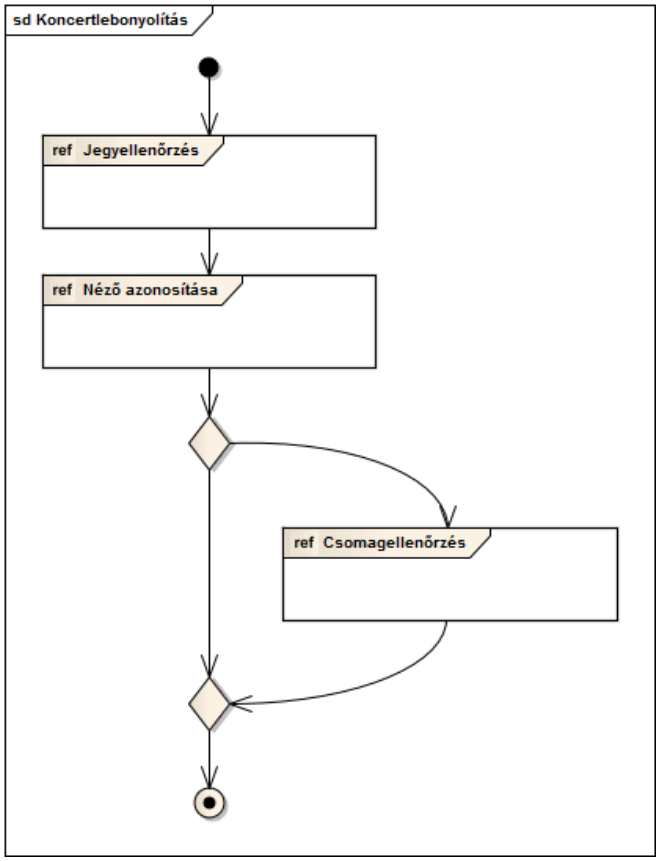
consider(P,Z): az ignore ellentettje, csupán a Z-beli üzeneteket veszi figyelembe az interakció során

assert(P): ezzel az operátorral biztosítható, hogy a P által specifikált lefutások pontosan az elvártak legyenek, és minden más lefutás nemkívánatos a rendszerben

neg(P): az operátorral a p operandus által specifikált interakciók nem megengedettek

Interakciók áttekintése

Nagyobb méretű, komplex interakciós diagramok esetén könnyen elveszhet az átláthatóság. A tevékenységdiagramok előnyeit kihasználva, lehetőségünk van az interakciók összekötésére, úgy mintha tevékenységek lennének egy tevékenységdiagramban. Az ábrán erre látunk példát.



87. ábra: Belépés megjelenítése interakciós átekintés formájában

Itt is megengedettek a tevékenységdiagramon megszokott elágazás és összeolvasztás, de kifejezetten csak zárójelezett formában. Csak informális alkalmazásra valók, mivel formális leírásuk még nem teljes.

10. Tervből kód: a leképezés fortélyai

A szoftvertervezés utolsó lépéseként az elkészített modellek kiválasztott programozási nyelven történő implementálás következik. E lépés részben automatizálható, azaz már a tervezés során célszerű olyan szoftverrel dolgoznunk, amely támogatja az automatikus kódgenerálást. Az alábbi táblázat néhány ilyen szoftvert említ:

A szoftver neve	Támogatott legfontosabb programozási nyelvek	Nyílt forrású?
ArgoUML	C++, C#, Java, PHP4, PHP5, Ruby	igen
BOUML	C++, Java, PHP, IDL, Python	nem
Dia	Python, C++, Javascipt, Pascal, Java	igen
Eclipse UML2 Tools	Java	igen
Enterprise Architect	C, C#, C++, Delphi, Java, PHP, Phyton	nem
MagicDraw	C#, C++, Java	nem
Microsoft Visio	-	nem
Netbeans	-	igen
Poseidon for UML	-	nem
Rational Software Architect	Java, C#, C++	nem
Umbrello UML Modeller	C++, Java, Perl, PHP, Python	igen
UMLet	-	igen
yED	-	nem

Támpontok osztályok leképezéséhez

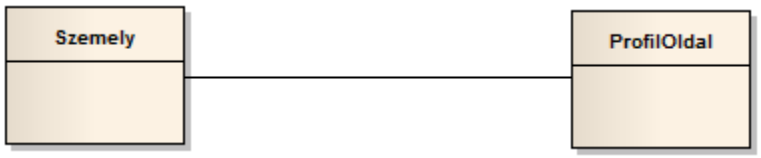
Ahogy azt az osztálydiagramot bemutató fejezetben ismertettük öt kapcsolattípus különíthető el. Ezek a következők voltak:

- függőség
- asszociáció
- aggregáció
- kompozíció

- öröklés

A függőség és az öröklés leképezése nem igényel különösebb magyarázatot, hiszen programozási nyelvtől függően az előbbi esetben include-álni, vagy importálni kell a megfelelő csomagokat, míg utábbi esetben a nyelv által megszabott szintaktikával kell a származtatást elvégezni. Az asszociáció, aggregáció és kompozíció azonban néhány szempontra figyelnünk kell.

Asszociácós és aggregációs kapcsolatot leggyakrabban referenciával kezelünk az érintett programozási nyelvben. Tekintsük elsőként az alábbi esetet:



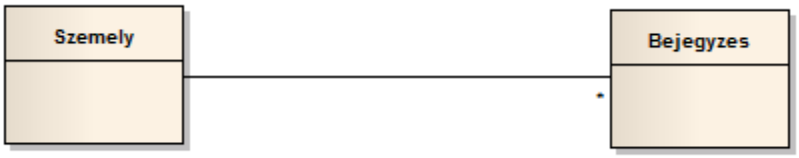
88. ábra: 1:1 multiplicitású asszociációs kapcsolat

Mint az látható a fenti osztálydiagram azt mutatja, hogy minden személynek pontosan egy profiloldala van, és minden profiloldal pontosan egy személyhez tartozik. Ez azt jelenti, hogy mind a személy objektumok, mind a profiloldal objektumok kölcsönösen küldhetnek üzenetet egymásnak. Ezt 1-1 referencia típusú adattaggal tudjuk biztosítani. C++ nyelven ez a következőképpen adható meg:

```
class Szemely
{
      ProfilOldal *m_ProfilOldal;
public:
      Szemely();
      virtual ~Szemely();
};

class ProfilOldal
{
      Szemely *m_Szemely;
public:
      ProfilOldal();
      virtual ~ProfilOldal();
};
```

Látható, hogy mindkét osztályban egy-egy pointer jött létre, melyek segítésével elérhetők a másik osztály metódusai. Kissé módosul a programkód a következő esetben:



89. ábra: 1:N multiplicitású asszociációs kapcsolat

A fenti ábra azt mutatja, hogy egy személy több bejegyzést tehet az oldalon, tehát módot kell adni arra, hogy egy személy több bejegyzését tároljuk, vagy elérjük. Éppen ezért a személy osztályban erre lehetőséget kell biztosítanunk. A kód a következő lehet:

```
class Szemely
{
      ProfilOldal **m_ProfilOldal;
public:
      Szemely();
      virtual ~Szemely();
};
```

De természetesen az is megfelelő lehet, ha egy konténer adatszerkezetet alkalmazunk a könnyebb kezelhetőség miatt:

```
class Szemely
{
      vector<ProfilOldal *> m_ProfilOldal;
public:
      Szemely();
      virtual ~Szemely();
};
```

Ezen megvalósítás a C++ nyelv STL vector adatszerkezetét alkalmazza, amely tetszőleges számú objektum tárolására alkalmas. Ahogy azt láthatjuk nem maguk az objektumok kerülnek tárolásra, hanem csupán azok pointerei biztosítva így a gyenge láncolást a két osztály objektumai között.

Aggregációs kapcsolat esetén sok esetben ugyanúgy járunk el, mint asszociációs kapcsolat meglétekor. A modellben jelezzük azt, hogy erősebb kapcsolat él a két osztály között, de programozási nyelvtől függ, hogy ezt implementációs szinten is meg tudjuk-e jeleníteni. C++ nyelven ez nem jelent változást: 1:1 kapcsolat esetén legtöbbször mutatóval, míg 1:N kapcsolat esetén pointerre mutató mutatóval, vagy valamilyen tároló adatszerkezettel dolgozunk.

Kompozíciós kapcsolat esetén – tekintettel arra - , hogy fizikai tartalmazás is fennáll, valamint, hogy az élettartam szempontjából is erős megkötéseink vannak a tartalmazó és a tartalmazott objektumokra

nem pointereket tárolunk, hanem konkrét objektumokat. Ez azt jelenti, hogy például a vector tároló hasonlóan definiálható:

```
vector<Igeny> Igenyek;
```

11. Irodalomjegyzék

R. Miles, K. Hamilton, Learning UML 2.0, A Pragmatic Introduction to UML, O'Reilly Media, 2006

H. Störrle, UML 2 – Unified Modeling Language, Panem Könyvkiadó, 2007

T. Tarczali, UML diagramok a gyakorlatban, Typotex, 2011