Írta:
TARCZALI TÜNDE

UML DIAGRAMOK A GYAKORLATBAN

Egyetemi tananyag

2011

LEKTORÁLTA: Dr. Nagy Zoltán, MTA Számítástechnikai és Automatizálási Kutatóintézet

TÁMOGATÁS:
Készült a TÁMOP-4.1.2-08/1/A-2009-0008 számú, „Tananyagfejlesztés mérnök informatikus, programtervező informatikus és gazdaságinformatikus képzésekhez” című projekt keretében.

ISBN 978-963-279-524-9

KÉSZÜLT: a Typotex Kiadó gondozásában

FELELŐS VEZETŐ: Votisky Zsuzsa

AZ ELEKTRONIKUS KIADÁST ELŐKÉSZÍTETTE: Benkő Márta

KULCSSZAVAK:

Unified Modeling Language, UML, modellezés, modellezési eszközrendszer, rendszertervezés, objektum-orientált tervezés, UML diagramok, OMG

ÖSSZEFOGLALÁS:

A tananyag az UML modellezési eszközrendszer használatának sokrétűségét vázolja fel egy gyakorlati példán keresztül. Mintapéldán át megmutatja az UML diagramjainak, táblázatainak használhatóságát a szoftverfejlesztési életciklus fázisaiban. Az anyag az UML-t már alapszinten ismerő hallgatóknak készült, az alapfogalmak bemutatása nem célja, inkább egy áttekintést szeretne nyújtani az UML alkalmazásának lehetőségeiről a diagramok és táblázatok használatának egy rendszertervezési projekt lépéseiben történő bemutatásával. Reményeim szerint ez jó alapot nyújt majd a hallgatóknak további tanulmányaik során, illetve a későbbiekben, tanulmányaik befejeztével a vállalati környezetben.

A tananyag az UML nyelv bemutatására szorítkozik, nem tartalmaz teljes bevezetést a rendszerfejlesztési módszertanokba illetve az objektumorientáltságba, bár körvonalaikban ezek is bemutatásra kerülnek.

A dokumentum gondolatainak alapját Harald Störrle UML 2 című könyvében valamint a Russ Miles és Kim Hamilton Learning UML 2.0 könyvében megismert fogalmak adják, és természetesen az OMG (Object Management Group) aktuális UML szabványa által meghatározott jelölésekkel találkozhatnak az olvasók.

Tartalom

1. Bevezetés

A tananyag célja, a hallgatók megismertetése az UML-ben (Unified Modeling Language) rejlő lehetőségekkel. Egy gyakorlati példán keresztül mutatja be az UML diagramjainak, táblázatainak használatát a szoftverfejlesztési életciklusokon keresztül. Az anyag az UML-t már alapszinten ismerő hallgatóknak készült, az alapfogalmak megismertetése nem célja, inkább egy áttekintést szeretne nyújtani az UML használatának lehetőségeiről egy projekt végigjátszásán keresztül.

A tananyag az UML nyelvre szorítkozik, nem tartalmaz teljes bevezetést a rendszerfejlesztési módszertanokba illetve az objektumorientáltságba.

A könyv anyagának gondolatainak alapját Harald Störrle UML 2 című könyvében valamint a Russ Miles és Kim Hamilton Learning UML 2.0 könyvében megismert fogalmak adják, és természetesen az OMG (Object Management Group) aktuális UML szabványa által meghatározott jelölésekkel találkozhatnak az olvasók.

2. UML általános áttekintése

Az UML története, céljai, elemei

A hetvenes évek elején jöttek létre az első szoftverfejlesztési módszertanok. A szoftverek bonyolultságának folyamatos növekedésével egyre inkább szükségessé vált az előzetes tervezés. A terület folyamatosan fejlődött tovább, így a kilencvenes évekre, már számos módszertan létezett. Ezek konkuráltak egymással, bár némely módszertanok között néha csak kevés eltérés volt tapasztalható.

A három fő irányzat vezetője, Jim Rumbaugh, Grady Booch és Ivar Jacobson 1994-től egyesítették erőiket, és közösen kezdtek el dolgozni. 1995-től a szabványosítási törekvéseket már az OMG (Object Management Group)[1] szervezte, és máig szervezi. Az UML jelenlegi verziója a 2.3.

2.1. ábra: Grady Booch, Jim Rumbaugh és Ivar Jacobson[2]

2.2. ábra: Object Management Group

Az UML egyes diagramjai nem tekinthetők újonnan definiált diagramoknak, nagyon sok közülük korábbi módszertanokból, vagy éppen más területekről átvett modellezési elem. Az idő-diagramokat például az elektronikai tervezésben már alkalmazzák, az osztálydiagramok pedig az ER-diagramokkal (entity-relationship) mutatnak rokonságot.

Az UML mint modellezési eszköz megjelenése a maga nemében azért volt jelentős, mert a korábbiakhoz képest egy szabványos, jól átgondolt, egymásra épülő és egymással kölcsön-hatásban lévő diagramokat és egyéb modellezési eszközöket nyújtó szabványként jelent meg,

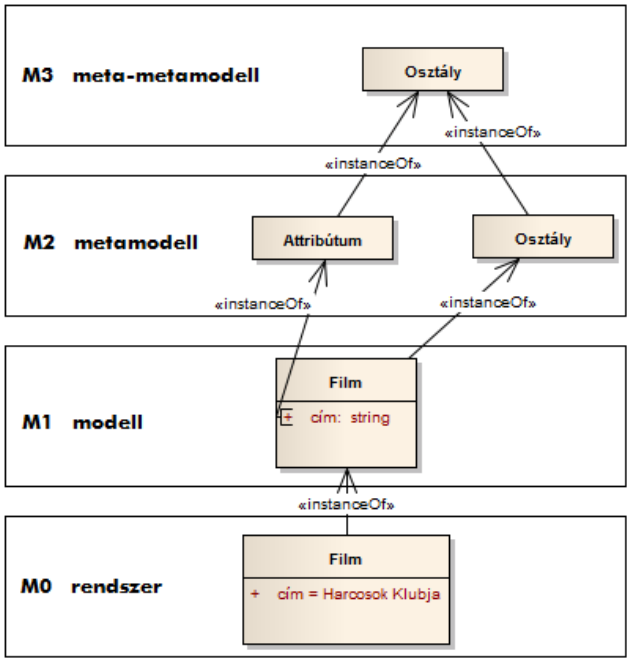
[1] Object Management Group - http://www.omg.org/
[2] http://www.sa-depot.com/?page_id=217

amelynek fejlesztése folyamatos, a szoftvertervezési igényekkel lépést tartva többrétegű rendszermodellek kialakítását teszi lehetővé.

Az UML nem módszertan, tehát nem adja meg a szoftverrendszerek tervezési és fejlesztési lépéseinek egyértelmű egymásutániságát. Eszköz, amely segítséget nyújt a modellezéshez. Ez a modellezés több szempontú lehet: függhet a megrendelőktől, a felhasználóktól, a modellező személyétől. A modellezési nyelv lehetőséget nyújt a különböző nézőpontok szerint kialakítani a modelleket, ezt támasztja alá az, hogy az egyes modellezési diagramok mellett például táblázatos formában is megjelennek a rendszerre jellemző információk.

Az UML felépítése

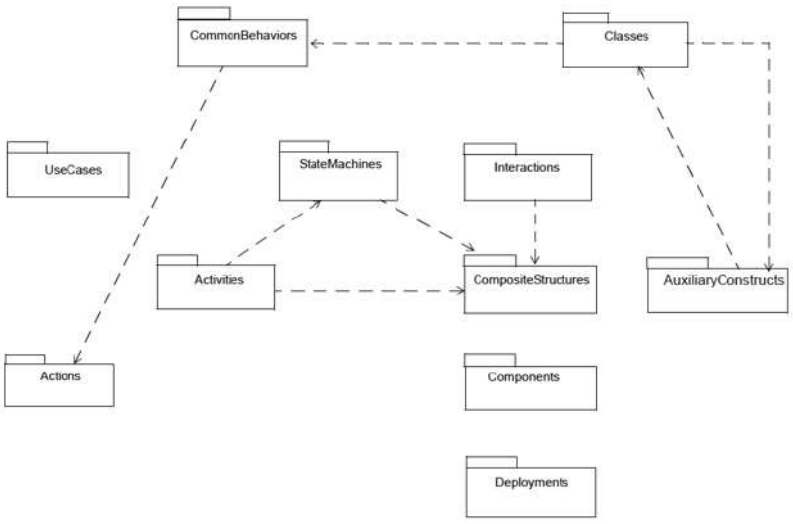
AZ UML felépítése egy metamodellezési nézőpontból indul ki. Ez azt jelenti, hogy egy rendszer egy modell példánya, a modell pedig egy metamodell (az UML metamodell) példánya, a metamodell pedig egy meta-metamodell példánya. Az UML metamodell architektúrája látható a 2.3 ábrán.



2.3. ábra: Az UML metamodell architektúrája

Az M0 szint osztályai és objektumai az M1 szint osztályainak, objektumainak a példányai, amelyek az M2 metamodell szint Attribútum illetve Osztály metaosztályainak példányai, ezek pedig az M3 szint Osztály metaosztályának példányai. Az architektúra tekinthető úgy is, mint a megvalósított futó rendszertől, mint az M0 szintről felfelé az M1 modell, mint Java-program szinten, majd M2 metamodell mint Java nyelvtan szinten keresztül eljutunk az M3 meta-metamodell szintig, amely egy formális nyelv.

Az M2 szinten belül az UML további részekre tagolódik: a magra (Infrastructure), magára a nyelvre (Superstructure) és a kiterjesztésekre (Profiles). Ezek a rétegek továbbtagolódnak szegmensekre, a szegmensek pedig csomagokra.



2.4. ábra: Az UML Szuperstruktúra csomagjai[3]

Az UML-ben többféle diagramtípussal találkozhatunk. A diagramok használhatók egy meglévő rendszer szemléltetésére modellezésére, vagy egy új rendszer megtervezésére. A diagramok mellett találhatunk táblázatokat, illetve szöveges formákat is. Erre azért is szükség van, mert az UML gyors fejlődését a modellezést támogató, úgynevezett CASE eszközök nehezen követik, így lehetőséget kell adni a más formában történő tervezésre is.

Több diagram is vonatkozhat ugyanazon modellre, a diagramok átfedhetik egymást. Azonban mindegyik típus eltérő nézőpontból mutatja meg a modellt. A diagramokat az UML-ben két csoportra oszthatjuk. Egyik csoportjuk a rendszert strukturális szempontból, másik viselkedés szempontjából vizsgálja, modellezi.

A struktúradiagramok központi diagramja az osztálydiagram. Minden egyéb diagram ebből származtatható: csomagdiagram, kihelyezési diagram, architektúradiagram, együttműködési diagram. A viselkedési diagramokhoz nem határozható meg ilyen módon egy „ősdiagram”. Találkozunk az UML-ben állapotautomata diagrammal, tevékenységdiagrammal és interakciós diagramokkal. Ez utóbbinak három típusa létezik: idődiagram, kommunikációs diagram és szekvenciadiagram. Egy speciális diagram még a viselkedési diagramokon belül az interakciós áttekintési diagram.

Ezek tehát azok a diagramtípusok, amelyekkel az anyagban találkozni fogunk, és megismerjük, hogyan használhatók rendszermodellezésre. Mindezt egy esettanulmány példaként való bemutatásával tesszük meg.

[3] http://www.omg.org/spec/UML/2.3/Infrastructure/PDF

Az UML 2.0

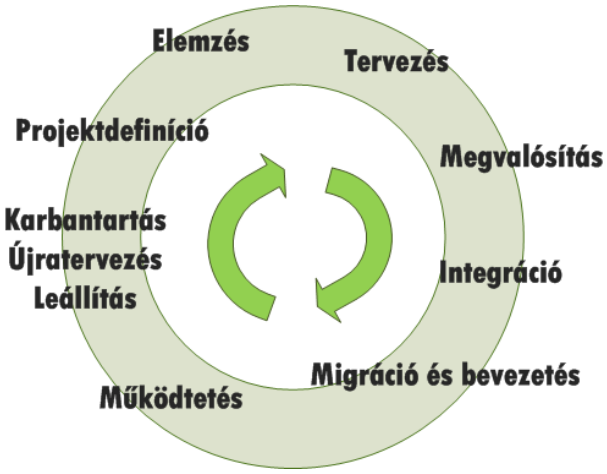
Az UML újabb verziójában több újdonsággal is találkozhatunk. A metamodell nagy változásokon ment keresztül, pl. a tevékenységek és az interakciók területén teljesen megújították. Számos új fogalom jött létre, így például a CompositePart a Port, a Connector, vagy az Interaction és Activity, illetve az ezekkel összefüggő fogalmak. Több új jelöléssel bővítették a modellezési lehetőségeket. Ezek közül vannak olyanok, amelyek új fogalmakat hordoznak (CompositePart, Connector) de olyanok is, amelyek egy régebbi jelölés újragondolásaként jöttek létre, és javítják a kifejezőerőt, vagy a modellezést egyszerűsítik.

Vannak még további tisztázatlan kérdések, de a nyelv folyamatos fejlesztésével (pl. az Infrastructure specifikáció az UML 2.3 verziójában) megválaszolásra kerülnek. Mindez nem azt jelenti, hogy az UML nem fejlődhet tovább, hiszen a szoftvertechnológia fejlődésével további nyitott kérdések merülnek fel. Így például a SysML kifejlesztése a rendszerek modellezéséhez is egy új eszköz a területen.

3. UML környezete

Szoftveréletciklus

A szoftverfejlesztés életciklusának ismertetésére csak röviden térek ki ebben a fejezetben, részletes bemutatása a Szoftvertechnológia tárgy feladata. Említeni azonban mindenképpen szükséges, hiszen az UML diagramok ezen életciklusok mindegyikében megjelennek, támogatják az egyes fázisok végrehajtását, illetve az egyes fázisokból történő továbblépést. A szoftverfejlesztés életciklusait mutatja be a 3.1. ábra:



3.1. ábra: A szoftver életciklusának fázisai

Ezt az életciklust szabvány definiálja [ISO 12207 (1995)], ebben a formában általánosan elfogadott. A szoftver életciklusának vagy a szoftverfolyamat szakaszait fázisoknak hívják.

Az UML diagramjai a gyakorlati szoftverfejlesztésben

Az UML a nevéből adódóan egy modellezési nyelv. A modellezés lehet egy eredeti szerkezetnek, folyamatnak vagy rendszernek a leképezése.

A szoftverfejlesztésben a modellezés az életciklus összes fázisában felbukkan, azonban inkább a kezdeti szakaszokra jellemző. Azért is fontos hangsúlyt fektetni a szoftverek tervezésére az implementációt megelőzően, mivel jó tervezéssel kardinális hibák küszöbölhetőek ki, és minél hamarabb felfedjük ezeket a hibalehetőségeket annál könnyebb és nem utolsósorban kevésbé költséges az eltávolításuk vagy a megelőzésük Ezért tehát érdemes időt szakítani az alapos tervezésre.

Ennek egyik eszköze lehet az UML. Eredetileg Unified Method-nak hívták, ezt a nevet azonban félrevezetőnek találták, hiszen ez nem egy módszertan, hanem egy eszközrendszer, így átnevezték Unified Modeling Language-re. Ahhoz hogy módszertanként szerepelhessen szükség lenne egy eszközrendszerre, illetve jelölésre és egy technikára. Technikát azonban az UML nem szolgáltat, teljesen módszersemleges.

A tervezőtől, modellezőtől függ, hogy mely diagramtípusokat alkalmazza a munkája során. Ez, mint már említetésre került korábban függ attól is, hogy az életciklus mely fázisához kötődik, kinek, kiknek kerül bemutatásra, kik fogják használni, milyen szokásjogok érvényesülnek a rendszerben.

A tananyag egy képzeletbeli projekten keresztül mutatja be az UML diagramok használati lehetőségeit A szoftverfejlesztési életciklus fázisai alatt az UML különböző diagramjaival találkozhatunk, különböző diagramtípusok használata célszerű. A tananyag egy esettanulmányon keresztül mutatja be ezeket a lehetőségeket. Az esettanulmány nem valódi esetet tárgyal, de gyakorlati példaként jól mutatja az UML eszközeinek alkalmazását. A hangsúly az elemzés és a tervezés fázisán van. Sajnos a tananyagterjedelme nem elegendő a teljes körű bemutatáshoz, de átfogó képet próbál adni a használat sokszínűségéről.

Esettanulmány

A projektet egy junior tanácsadó szemszögéből tekintjük. A cél egy képzeletbeli jegyiroda és szervezőiroda, a TicketActual informatikai rendszerének átszervezése, megújítása. Az iroda célja az átalakítással a költségek csökkentése a megfelelő munkafolyamatok automatizálásával.

A cég természetesen rendelkezik már informatikai rendszerrel, az új rendszernek tehát ehhez kell kapcsolódnia. Ilyen meglévő részrendszer az „BonusPoints” program, amely többek között a felhasználók által elért bónuszokat tartja nyilván, illetve a partneradatokat a „BestParts” rendszer kezeli.

A vállalat vezetősége felméréseket végeztetett és ez alapján úgy döntöttek, hogy a költségek leginkább a jegyfoglalás és a bejelentkezés területén csökkenthetők az informatikai hálózat szélesítésével. Az új rendszernek az „TicketFirst” nevet adták.

Elemzés

Egy projekt lefutása során az első feladat a feladat definiálása. Ennek során szükség van a projektben résztvevők megismerésére, rendszer környezetének felmérésére, a feladat leírására. Ez tulajdonképpen nem más, mint információgyűjtés arról, hogy milyen résztvevői lesznek magának a projektnek, mi az pontosan, amit meg kell valósítani. Az információgyűjtéshez leginkább az adott szakterületen dolgozó résztvevők használhatók. Velük úgynevezett üzleti interjúkat kell készíteni, amelyeknek a strukturált szöveggé történő átalakításával a szakterület üzleti folyamatai leképezhetőek. Ennek természetesen előfeltétele, hogy szakterületi fogalmakkal is meg kell ismerkedni. Az üzleti folyamatok a dokumentumok alapján leltárba rendezhetőek, valamint táblázatos séma alapján a részleteiket ki lehet fejteni.

Egyes esetekben lehetséges, hogy a projekt mérete nem igényli az üzleti folyamatok azonosítását. Kisebb projektek esetében elégséges a használati esetek leltárba vétele, illetve táblázatos formában részleteik kifejtése. Nagyobb projektek esetében a használati esetek, mint az üzleti folyamatok finomításai jelennek meg.

A rendszer dinamikus modellezését az üzleti folyamatok és használati esetek azonosítása után a tevékenységdiagramok és állapotautomaták konstruálásával lehet folytatni. Ezek a diagramok a használati esek és az üzleti folyamatok lépéseinek lefutását mutatják meg. A rendszer és a környezete közötti interakciók leírására az interakciódiagramok használhatók. Az interakciódiagramok közül az esettől függően választhatunk. Előnyeiket és hátrányaikat az adott fejezetben tárgyalom. Az eddig leírt követelmények a rendszerrel kapcsolatban a funkcionális követelményeket fedik le. A nemfunkcionális követelmények leírására használhatók például a szakarchitektúra diagramok, de ezek szövegesen is rögzíthetők.

Az eddig említett eszközök a rendszer folyamatainak leírására használatosak. A strukturális modellezést segíti a már megismert szakterületi sajátosságok leírására használt szakterületi modell. Ehhez kapcsolódóan elkészíthető az elemzési osztálydiagram, illetve a fogalmak összegzésére használt fogalmi szótár. A szakterületi objektumok viselkedése objektum-életciklusokkal vagy osztályinterakciókkal írható le

Tervezés

A tervezés és az elemzés fázisa általában nem különíthető el élesen egymástól, ez elemzés gyakran átfolyik a tervezésbe. A tervezés során már nem csak azt azonosítjuk, hogy mit modellezünk, hanem kitérünk arra is, hogy hogyan fogjuk a megvalósítást elvégezni. Ebben a szakaszban azonban még nem foglalkozunk technológiai részletekkel.

A tervezés elején azonosítani kell a szoftverarchitektúrát. Itt meg kell adni, hogy milyen alrendszerei, komponensei legyenek a rendszernek. AZ UML erre többek között a rendszer-montázsdiagramot használja. Másik fontos diagramtípus a rendszerarchitektúra, amely megmutatja, hogy az egyes alrendszerek és komponensek hogyan helyezkednek majd el a rendelkezése álló hardvereken.

A szoftverarchitektúrát több lépésben lehet finomítani, erre az UML csomagdiagramja, illetve a rendszer-montázsdiagramja ad lehetőséget. Lehetőség van a felhasználói interfészeken lefutó interakciók modellezésére is.

A tervezési szakasz utolsó lépésében kerül sor a tervezési osztálydiagram megrajzolására, amellyel az objektumok finomabb felépítése határozható meg. Ezekhez kapcsolódóan az objektum életciklusok és objektuminterakciók kidolgozása is megtörténhet.

Megvalósítás

Az elemzés és tervezés után következik a megvalósítási fázis. Elméletileg erre a fázisra már minden szakmai kérdés tisztázásra kerül. Itt a megvalósítás technológiai részleteire kell összpontosítani. Ezt a fázist manapság leginkább a programozás határozza meg. A programozás lépése ma már automatikus kódgenerálással is elérhető. A megvalósítási osztálydiagramok például tekinthetők a kódgenerálás alapjának, de a tevékenységi diagramokkal megadott algoritmusokból is kód állítható elő.

A kódból való visszaalakítással a rendszer viselkedése és rendszerstruktúrája is leírható. Hasznos lehet például hibakeresés céljából. Ebben a lépésben az osztálydiagramok és az objektuminterakciók a leginkább használhatóak.

Integráció

Az integráció során a már meglévő modulokat integrálják, futtatható egységgé illesztik össze. A szoftverfejlesztési fázis során az objektum- és rendszer-montázsdiagramok alkalmazhatóak. A tárolási struktúrák megjelenítését a csomagdiagramokkal modellezik. A célkörnyezet a telepítési és kihelyezési diagramokkal modellezhető.

Bevezetés és migráció

A rendszer használata nem akkor kezdődik el amikor teljesen elkészül. A használatot megelőzi a rendszer tesztelésének fázisai, pl. a rendszertesztek, a kézikönyvek, felhasználói dokumentáció elkészítése, a későbbi betanítási folyamat megkönnyítésére, a hardverek, szoftverek beszerzése, illetve a már meglévő rendszerbe való beillesztés.

Mindezek támogatásához hasznosak lehetnek a dialógusok lefutásának tervei, az adatmodellek és a folyamatlefutások. A rendszerbe történő beillesztéshez és a telepítéshez használhatóak a kihelyezési modellek.

Ha egy régi rendszerről állunk át egy új rendszerre, akkor az adatok migrációjához az említett összes diagram felhasználásra kerülhet.

Működtetés és karbantartás

A rendszer használata során nem merül fel a korábban használt diagramok szükségessége. A karbantartási fázisban viszont az összes készített modellre szükség lehet a javítandó hiba tulajdonságaitól függően. Pl. egy programkódbeli hiba javításához leginkább a megvalósítási osztálydiagram vagy objektumdiagram lehet a programozók segítségre.

Újratervezés, leállítás

Az újratervezés vagy leállítás kérdésének eldöntése egy meglévő rendszer esetében alapos megfontolást igényel. Meg kell nézni, hogy megéri e a rendszer átalakítani, vagy egy új rendszer kifejlesztése már költséghatékonyabb megoldást nyújt, esetleg nem lehetséges a régi rendszer továbbfejlesztése a felmerült igényeknek megfelelő módon.

Újratervezés esetén a meglévő modellek felhasználása sok segítséget nyújt. Szinte az összes modell felhasználható ebben az esetben.

4. 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.

4.1. táblázat: Az üzleti folyamatok és a használati esetek jellemzői

Kritérium	Üzleti/Rendszerfolyamat	Használati eset
Tartalom	komplex szakterületi/ technikai lefutás	egyetlen szakterületi/technikai munkalépés
Mértékek	mérhető értéke vagy költsége van	ráfordítást igényel és erőforrásokat emészt fel
Kölcsönös viselkedés	használ/tartalmaz szolgáltatásokat és funkciókat	üzleti folyamatok részegysége
Finomítható	igen, funkciókon és szolgáltatásokon keresztül	funkció: igen, funkciókon és szolgáltatásokon keresztül szolgáltatás: nem
Megvalósító	szervezet vagy több rendszer és személy	egyetlen alkalmazás
Időtartam	hosszú (hetek, hónapok)	rövid, egyetlen perc, néhány másodperc
Megszakítás	lehetséges	automatikusan fut és nem szakad meg
Rendszer-, ill. alkalmazáshatárok	átlépi a határokat	definiálva van
Lefutás	esetleg csak részben automatikus	egy rendszer valósítja meg teljesen automatikusan, nem tartalmaz manuális közbenső lépéseket
Célcsoport	szakterület, szakemberek	technikai személyzet, fejlesztők
Nézőpont	az aktorok és alkalmazások alkotta rendszerbe befelé (fehér doboz nézet)	kívülről, egy rendszerre vagy alkalmazásra vonatkozóan, (fekete doboz nézet)

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.

Az üzleti folyamatok rendszerek felett állnak, a használati eset viszont egy rendszerre vonatkozik. A használati esetek rövid idő alatt és különösebb működés közbeni interakció nélkül futnak le. Az üzleti folyamatok megszakíthatóak, gyakran napokig vagy hónapokig zajlanak.

Ezek a feltételek nem mindig határozzák meg egyértelműen, hogy használati esetről vagy üzleti folyamatról van e szó. Előfordulhat, hogy egy konkrét esetben nem mindegyik teljesül egyértelműen. Ilyenkor a modellező feladata eldönteni, hogy melyikről van szó.

Szöveges folyamatleírás, interjú

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.*

Hierarchikus rendezés után az interjú strukturált szövegként történő megjelenítse így néz ki (4.2. táblázat):

4.2. táblázat: A koncertlátogatás interjú leírása strukturált szövegként

Ki	Milyen gyakran	Mit	Megjegyzés
-	1	Koncertlátogatás	
-	1	Jegyvásárlás	Online
N, TF	1	Felhasználó bejelentkezése	Login, Password
N, TF	1…n	Koncert kiválasztása	
N	1	Koncert adatainak megadása	Dátum (-tól -ig), város, helyszín
TF	1	Koncertek mutatása	időpont szerinti sorrendben
N	1	Koncert kiválasztása	
TF	1	Lehetséges helyek mutatása	Álló, ülő, sor, szék, jegyár
N	1…n	Hely kiválasztása	Egymás után több is lehetséges
TF	1	Koncertjegyek mutatása	(Alternatívák mutatása)
N	1	További adatok megadása	Elektronikus jegy, hagyományos jegy
N	1	Vásárlás megerősítése	
N, TF	1	Jegyek kifizetése	
N	1	Fizetés típusának kiválasztása	Hitelkártya, számla
N	1	Fizetési adatok megadása	
N	1	Fizetés megerősítése	
TF	v	Jegyvásárlás felvétele	
TF	0…n	Jegy nyomtatása	
-	1…n	Lebonyolítás	
Sz/N, BLA		Belépés	
Sz/N, BLA	1	Jegyellenőrzés	Automata vagy Személyzet segítségével
BLA	1	Jegyek ellenőrzése	Helyszín és dátum/ idő alapján
BLA	1	Jegykezelés	
Sz/N, BLA	1	Néző azonosítása	Bónuszkártya, hitelkártya alapján
BLA	1	Forgókapu nyitása	
-	1	Beléptetés	
EK/Sz		Csomagellenőrzés	Ellenőrzőkapu vagy Személyzet segítségével
N, Sz/BA	1	Forgóajtó nyitása	
TF	1	Bónuszjóváírás	
TF	0…1	Bónuszpontok jóváírása	
TF	0…1	Néző felértékelése	

Használati esetek és üzleti folyamatok: a modellezés követelményei

A rendszer működésének határai az üzleti folyamatok 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 az üzleti folyamatra hatással vannak. Ilyen üzleti folyamatnak tekinthető az előző fejezetben tárgyalt „Koncertlátogatás” folyamata. A kritériumok közül a következőket valósítja meg: több szakasza és lépése van; szüneteket tartalmaz

(jegyvásárlás és koncert lebonyolítása között), így hetekig, hónapokig eltarthat; több résztvevője van (néző, lebonyolító személyzet, a TF rendszer foglalási és könyvelési alrendszerei…); bizonyos részei automatikusan bonyolódnak le (pl. koncerthallgatás); mind az nézők, mind a jegyiroda számára a folyamat konkrét és mérhető hasznot illetve költséget realizál.

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

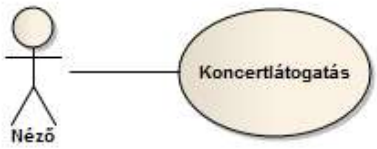
Az itt látható táblázatos forma egy lehetőség az üzleti folyamatok szabványos leírására. Az irodalomban ez többféle módon megjelenhet, a főbb jellemzői viszont azonosak az egyes táblázattípusoknak. A táblázat mezőinek jelentése a 4.3 táblázat alatt látható.

4.3. táblázat: A koncertlátogatás üzleti folyamat táblázata

<table>
<tr><td>Azonosító
ÜF - KL - TF</td><td colspan="2">Név
Koncertlátogatás</td></tr>
<tr><td colspan="3">Rövid leírás
Koncertlátogatás teljes folyamat a jegyvásárlástól a bónuszpontok felírásáig</td></tr>
<tr><td colspan="3">Érintett aktorok
1. Néző, 2. TicketFirst</td></tr>
<tr><td colspan="3">Kiváltó esemény
A néző el szeretne menni egy koncertre</td></tr>
<tr><td colspan="3">Előfeltétel
nincs</td></tr>
<tr><td colspan="2">Standard lefutás
1. Jegyvásárlás
2. Koncert lebonyolítása
3. Bónuszpontok jóváírása</td><td>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</td></tr>
<tr><td colspan="3">Utófeltétel
Mindhárom folyamat utófeltételeinek együttese</td></tr>
<tr><td colspan="3">Eredmény
A koncert lezajlik, az irodának bevétele származik belőle</td></tr>
<tr><td colspan="3">Gyakoriság
napi 3000</td></tr>
<tr><td colspan="3">Utalások
későbbiekben kerül kitöltésre</td></tr>
<tr><td colspan="3">Megjegyzések, nyitott kérdések
nincs</td></tr>
</table>

- A táblázat azonosítója egy egyedi azonosító, amelyre a további dokumentációban utalni lehet.
- A név az üzleti folyamat neve. Fontos a konzisztencia megőrzése az egész dokumentumon keresztül. Ne hivatkozzunk ugyanazzal a névvel két eltérő folyamatra vagy objektumra.
- A rövid leírás a folyamat összefoglalását tartalmazza
- Az érintett aktorok között megkülönböztetünk elsődleges és másodlagos aktorokat. Elsődleges aktornak az tekinthető, aki a főszerepet játssza a folyamat során. Jelen esetben ez a Néző lesz, aki elindítja a folyamatot.
- A kiváltó esemény egy olyan esemény, amelynek hatására a folyamat elindul. Itt például a kérés vagy igény, hogy a Néző szeretne egy koncertre ellátogatni.
- Előfeltétel a rendszer állapotára vonatkozó feltétel, amelynek teljesülnie kell ahhoz, hogy az üzleti folyamat gond nélkül lefusson. Ha ez nem teljesül, akkor az utófeltételek és a sikeres lefutás nem biztosítható.
- A standard lefutás tartalmazza azokat a lépéseket, amelyek a sikeres lefutáshoz kellenek. Elsődleges forgatókönyvként is emlegetik, ez a leggyakrabban előforduló eset.
- A kivételek és alternatívák tartalmazzák azokat a lehetőségeket, amelyek „nem férnek bele” a standard lefutásba, tehát leginkább valamilyen ettől eltérő lehetőségről beszélhetünk. Másodlagos forgatókönyvként is emlegetik.
- A rendszer állapotára vonatkozó feltétel, amelynek teljesülnie kell, ha a forgatókönyvek valamelyike lefut.
- Eredményként a résztvevők számára érzékelhető vagy megjelenő eredményt tekintjük.
- A gyakoriság az üzleti folyamat lefutásának gyakorisága.
- Utalások mezőbe kerülnek azon dokumentumok azonosítói, amelyek kapcsolatban állnak az adott üzleti folyamattal. Ilyenek lehetnek a felhasználói felület terve, vagy előzetes megkötések.
- A megjegyzések mezőbe bármi egyéb kerülhet ami az előbbiekbe nem fér bele.

Az aktorok és az üzleti folyamat kapcsolatát vizuális megjelenítéssel is megmutathatjuk. A 4.1-es ábrán látható a Néző aktor kapcsolata a tárgyalt üzleti folyamattal.

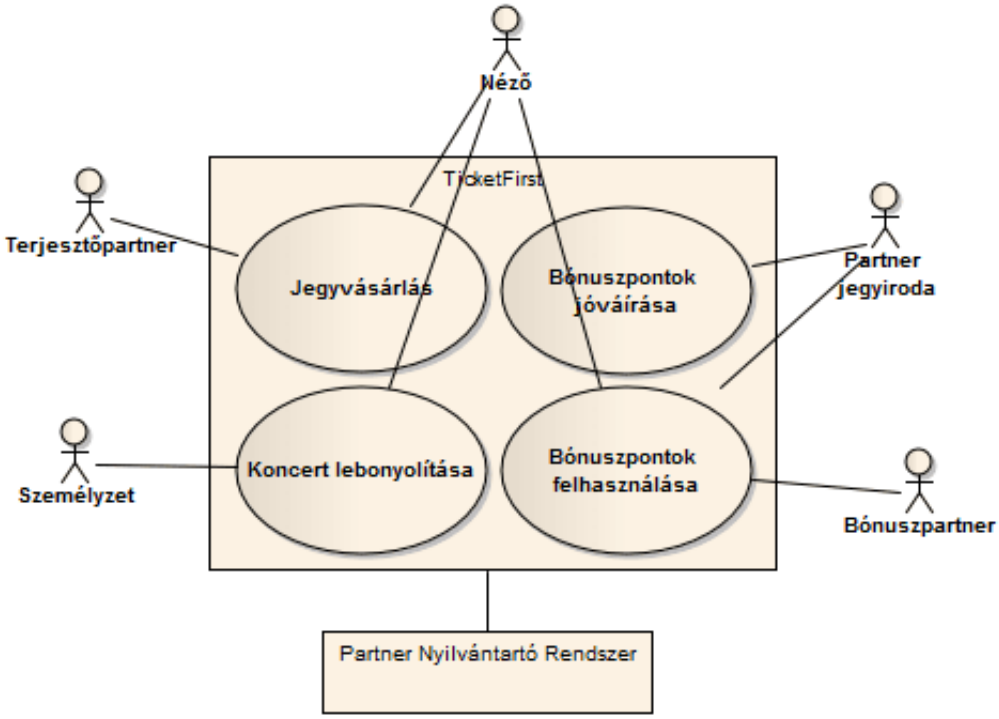


4.1. ábra: A koncertlátogatás üzleti folyamat és a néző aktor kapcsolatának megjelenítése

Az ábrán látható hogy az üzleti folyamat egy ellipszisben jelenik meg, az aktor pálcikaember formát ölt, a köztük lévő kapcsolatot pedig egy összekötő jelzi.

Üzleti folyamat leltár

Az üzleti folyamatok felsorolásával tehát meghatározhatók a rendszer határai. A folyamatleltárral vizuálisan is érzékeltethető a rendszer és környezetének elkülönítése. A 4.2-es ábrán látható a TF rendszerhez tartozó folyamatleltár.



4.2. 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:

4.4. táblázat: A jegykezelés használati eset táblázata

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	

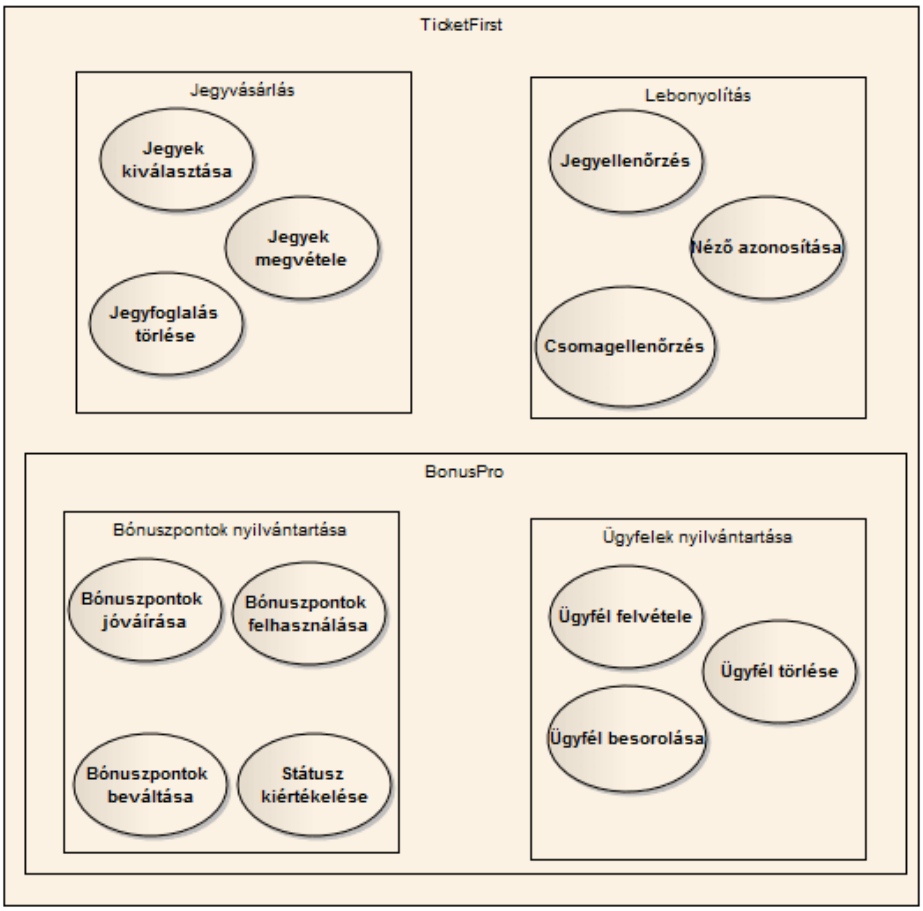
A használati eset táblázat nem sokban különbözik az üzleti folyamat táblázattól. Eltérés a következőkben van:

- 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.



4.3. ábra: A TicketFirst rendszer használati eseteinek leltára

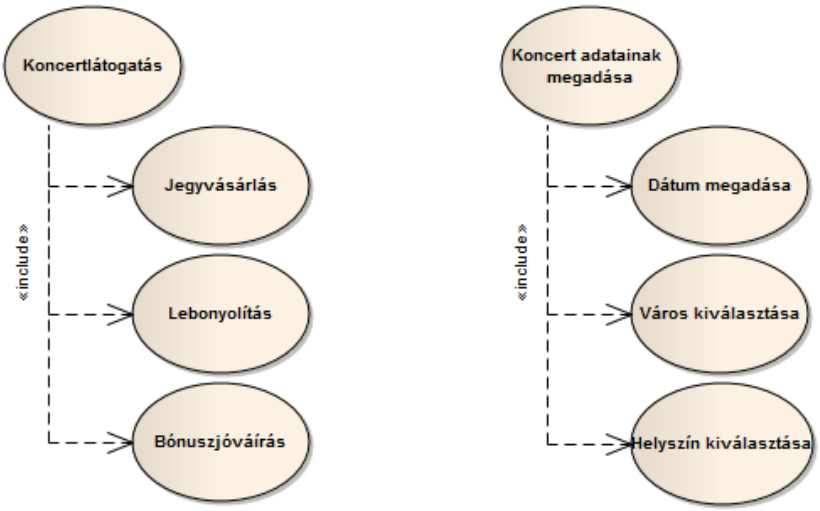
Észrevehető, hogy a kialakítandó rendszer tervezése során ahogyan lépünk előre, annál többet tudunk meg a funkcionalitásokról. Az üzleti interjúban még nem szereplő részetekre a modellezés során derül fény, így a résztvevőkkel több alkalommal is szükség lehet egyeztetésre. Természetesen nem azonnal tudunk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

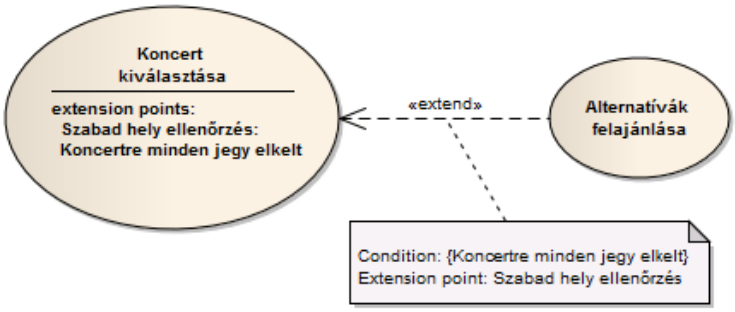
A 4.4. ábrán látható a Koncertlátogatás üzleti folyamat és a részét képező használati esetek magában foglalás relációja. A belefoglalt esetek magukban is értelmesek, a folyamat közben többször is lefuthatnak, de legalább egyszer szerepelniük kell. Így a dátum megadása többször is előfordulhat a koncert adatainak megadása közben, ha a Néző több időpontra kíván jegye foglalni, esetleg ő is szeretné megnézni a koncertet, de ajándékba is szán jegyet.



4.4. ábra: Include kapcsolat a folyamatok között

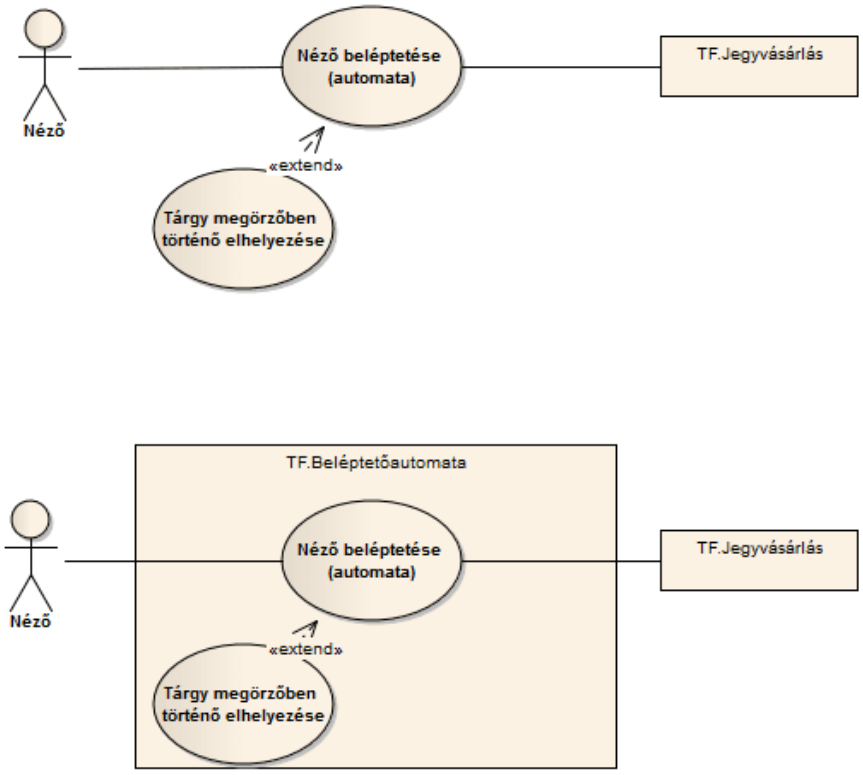
Kiterjesztés – extend

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 (4.5. ábra).



4.5. 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:



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

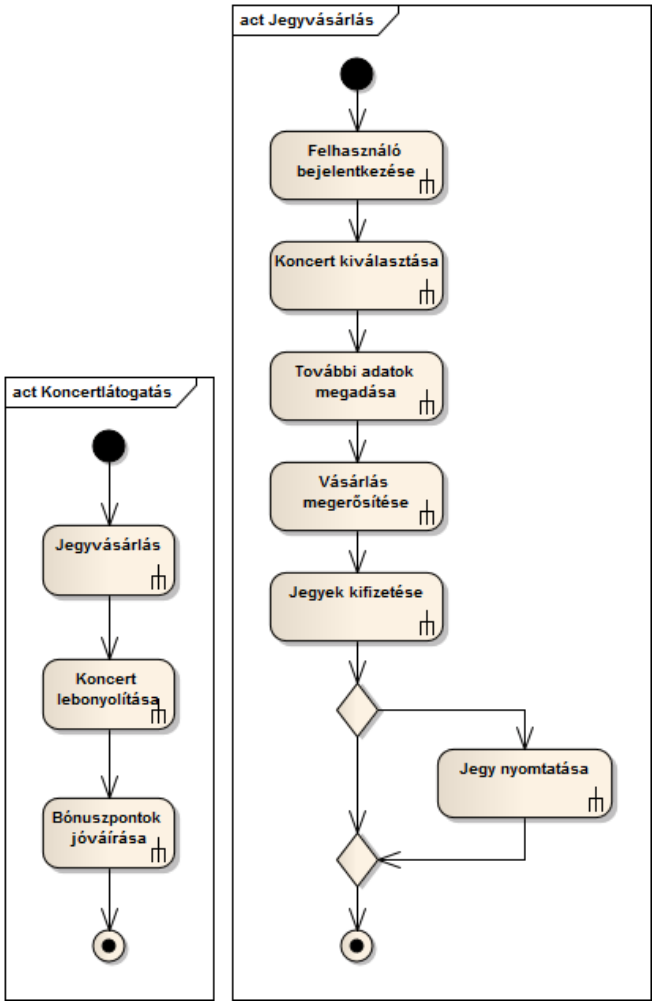
Látható, hogy a Néző beléptetése folyamat a Tárgy megőrzőben történő elhelyezésébe használati esettel van kiterjesztve. Kapcsolódik hozzá még egy Néző aktor és a TF.Jegyvásárlás alrendszer. Az alsó ábrán feltüntetésre került az alrendszer is ami a folyamatot megvalósítja.

5. A rendszer folyamatainak modellezése

A rendszerek folyamatainak leírására az egyik lehetséges diagramtípus a tevékenységdiagram. Az UML 2 tevékenységek szemantikája a Petri hálókon alapul. A ’70-es évektől kezdődően az adatfolyam-diagramok is a folyamatmodellezés kelléktárába tartoznak. Mindegyikük leginkább a szoftverfejlesztés elemzési fázisban használható.

Tevékenységdiagramok

A tevékenységdiagramok tevékenységekből, és a köztük kapcsolatot teremtő vezérlési folyamélekből állnak. A tevékenységeket lekerekített sarkú téglalappal jelöljük. A kezdőállapot, ahogyan az alábbi ábrán is látható, egy teli körként, a végállapot pedig kettős körként jelenik meg. A tevékenységek közötti egymásutániságot az irányt is mutató összekötőkkel jelöljük, ez mint nyíl jelenik meg az 5.1-es ábrán.

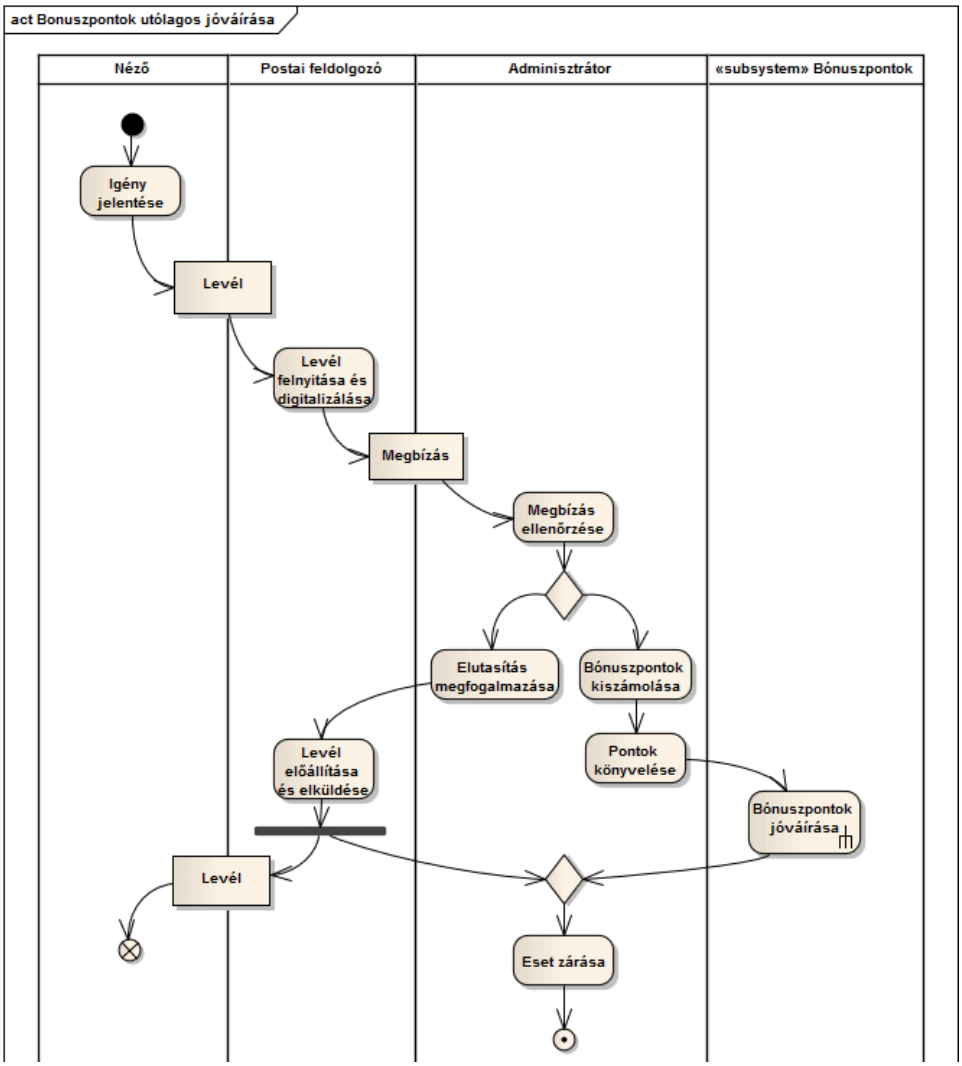


5.1. ábra: Koncertlátogatás és Jegyvásárlás folyamatok tevékenységei

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.



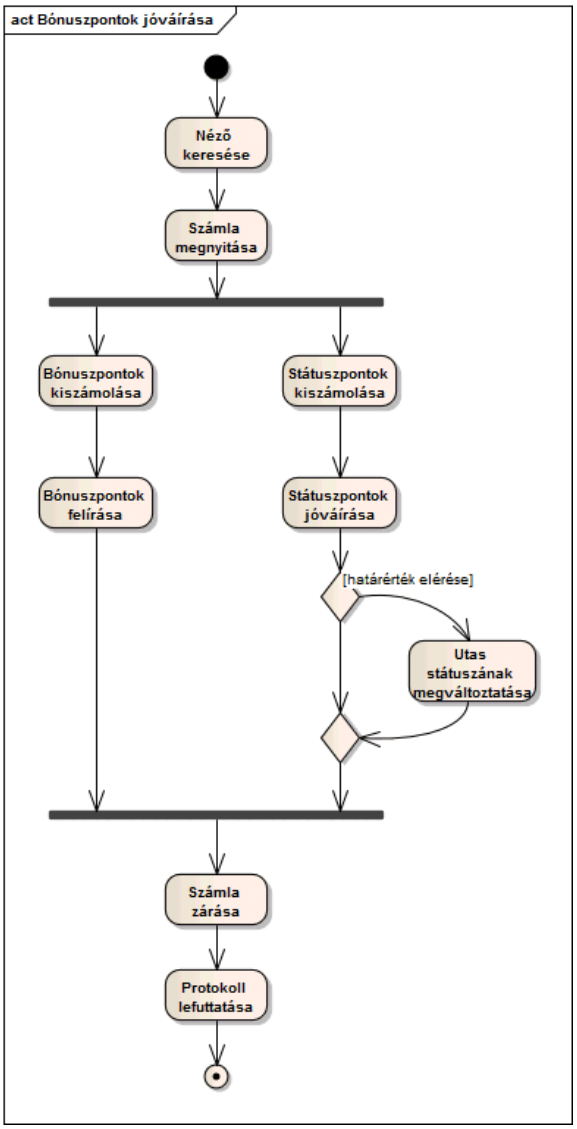
5.2. ábra: Bónuszpontok utólagos jóváírása

Az 5.2-es 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 folyam-vég pontban végződik, amely vezérlőjeleket fogad, de egyéb kihatásuk nincs a folyamatra. A végállapot ezzel szemben 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

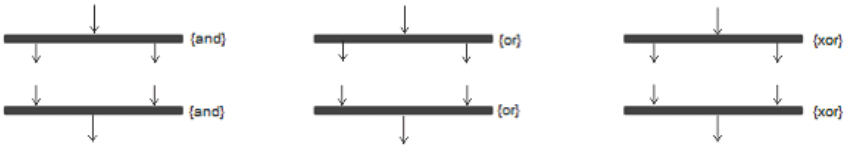
Az utólagos bónuszpont jóváírás üzleti folyamat lefutásának egyes lépései mind használati esetek. A Bónuszpontok jóváírása funkcionalitást a BonusPoints alrendszer valósítja meg. Ez a használati eset részletezhető egy tevékenységdiagrammal.



5.3. 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.

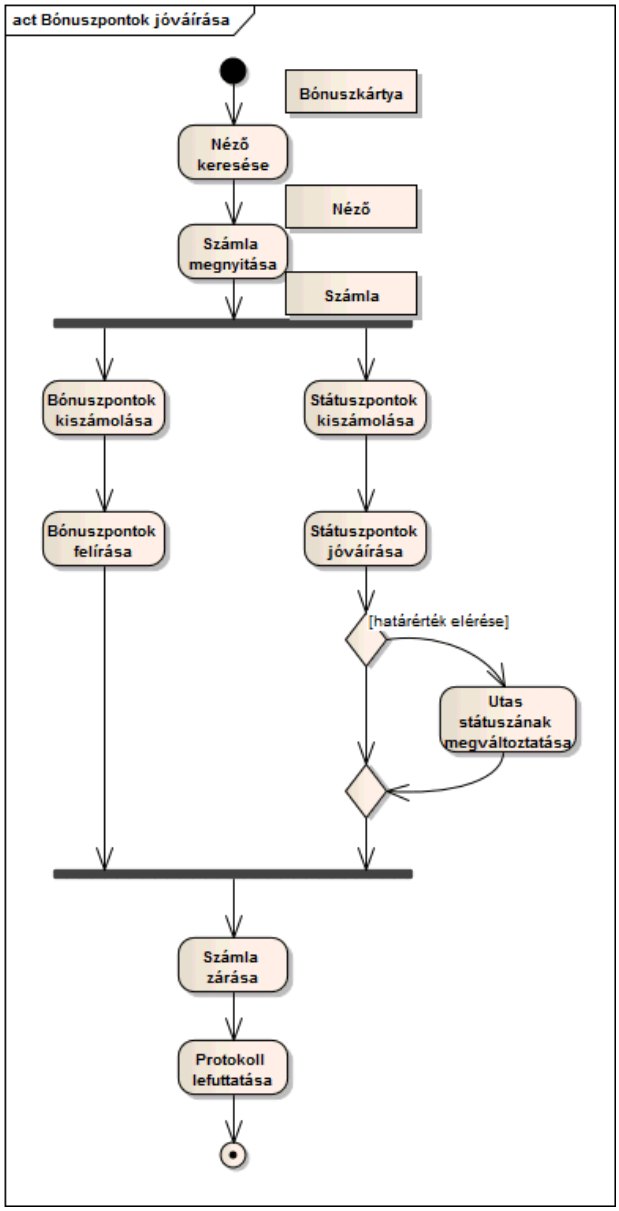
Az elválasztás és az összeolvasztás logikai feltétellel is ellátható, erre látunk példát az alábbi ábrán:



5.4. ábra: Elválasztás és összeolvasztás logikai feltételekkel

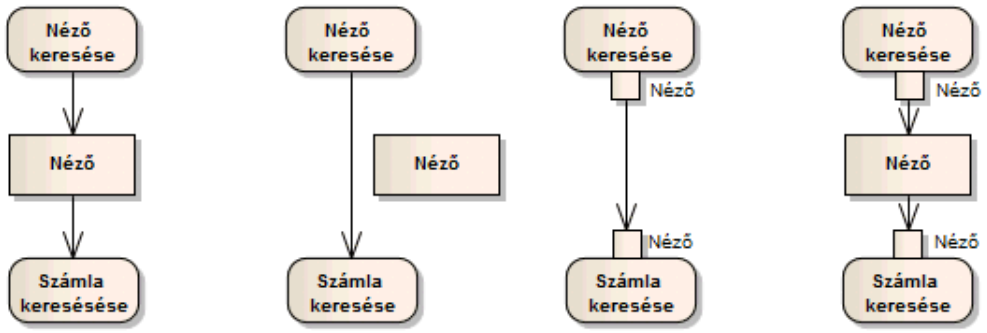
A korábbi UML verziókban az elválasztás és az összeolvasztás csomópontoknak zárójel-struktúrában párosan kellett megjelenniük, ez az UML 2-től kezdve már nem kell hogy így legyen.

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



5.5. 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.



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

Objektumfolyam csomópontok

Az objektumfolyam csomópontok mint depók képzelhetőek el, amelyekbe vezérlőjelek lépnek be és lépnek ki az objektumfolyam éleken keresztül. A csomópontok ezen tulajdonságuk alapján többféle paraméterrel rendelkezhetnek:

- Típus: megadható, hogy a csomópont csak egy bizonyos típusú osztály objektumait fogadja
- Állapot: a típus mellett megadható, hogy csak bizonyos állapotban lévő objektumok legyenek elhelyezhetőek az objektumfolyam-csomópontban
- Upperbound: ezzel a tulajdonságértékkel megadható, hogy hány vezérlőjelet tud fogadni a csomópont, vagyis a kapacitása, amely felett már blokkolja az objektumfolyamot. Alapesetben végtelen kapacitással rendelkeznek
- Ordering: ezzel megadható a vezérlőjelek befogadásának sorrendisége:
    - Undordered: nincs meghatározott sorrendiség
    - Ordered: speciális, meghatározott sorrendiség, amelyet egy viselkedésmodell felírásával lehet specifikálni egy feltételben
    - LIFO: a fogadott vezérjeleket a verem elv alapján tárolja
    - FIFO: a fogadott vezérjeleket a sor elv alapján tárolja

Ezek a sorrendiségek addig érvényesek, amíg a kimenő folyamél más sorrendiséget nem határoz meg.

Az általános objektumfolyam-csomópontokon kívül az UML két speciális alesetet is meghatároz. Az egyik az adatpuffer, amely egy olyan tranziens tároló, amelybe az adatelemeket

időlegesen tárolják és a rájuk való hivatkozás törli őket a pufferből. A másik az adattár, amely az adatok állandó tárolására szolgál. Minden bejövő adat rögzítésre kerül, nem törlődik kiolvasáskor.

5.7. ábra: Adattárak és adatpufferek jelölése az UML diagrammokon

Objektumfolyam-élek

Az objektumfolyam-élek a csomópontok és a tevékenységek közötti összekötők. Vezérjeleket hívnak le vagy vezérjeleket helyeznek el a tárolókban. Az éleknek is lehet tulajdonságokat adni a csomópontokhoz hasonlóan:

<<transformation>>: a vezérjelek lehívása vagy elhelyezése egybeeshet a vezérjelek átalakításával

Weight: megadható azon vezérjelek száma, amelyek a kapcsolódási folyamat során szükségesek. Ha egy élen kevesebb vezérjel jelenik meg, akkor a kapcsolódás nem biztosított

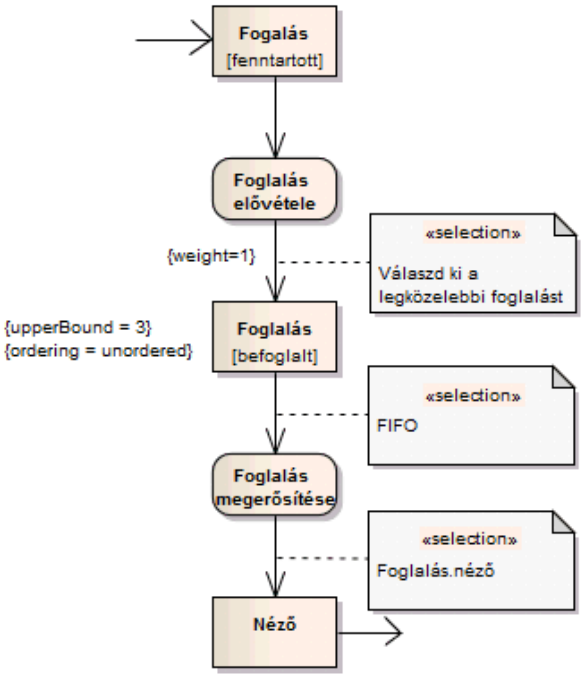
<<selection>>: megadható az élekre olyan sorrendiség, amely megmondja, hogy a vezérjeleket milyen sorrendben kell lehívni a csomópontból:

Unordered: nincs meghatározott sorrendiség

Ordered: speciális, meghatározott sorrendiség, amelyet egy viselkedésmodell felírásával lehet specifikálni egy feltételben

LIFO: a fogadott vezérjeleket a verem elv alapján lehet lehívni a csomópontból

FIFO: a fogadott vezérjeleket a sor elv alapján lehet lehívni a csomópontból

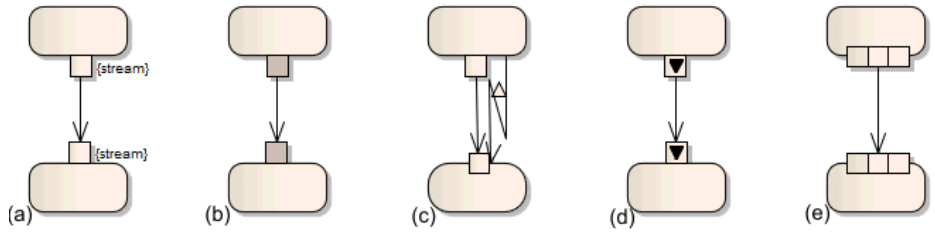


5.8. ábra: Tevékenységdiagram részlet objektumfolyam csomópontokkal és objektumfolyam élekkel

Az 5.8-as ábrán a Foglalás elővétele és a Foglalás megerősítése tevékenységek között találjuk a Foglalás osztály különböző állapotait, mint objektumfolyam-csomópontokat. A legelső csomópontba az osztály fenntartott állapotban lévő objektumai kerülhetnek be. A Foglalás elővétele egy objektumot helyez a Foglalás osztály befoglalt állapotú objektumába, azt, amelyiknél a foglalás dátuma a leghamarabbi. A csomópont kapacitása 3, tehát ennél több objektum nem kerülhet bele. A csomópontban a vezérjelek sorrendisége nincs meghatározva, de a kimenő objektumfolyam-él ezt FIFO sorrendiségre változtatja. A Foglalás megerősítésénél a Néző, aki számára a foglalás szól, azonosításra kerül és a Néző objektumfolyam-csomópontba kerül.

Szolgáltatáskomponensek

A rendszerben megjelenő funkcionalitásokat, mint a rendszer által nyújtott szolgáltatásokat is tekinthetjük. Ezek az egyes szolgáltatásokból - mint egymás utáni tevékenységekből - folyamokat lehet kialakítani. A köztük lévő kapcsolatokat pedig a szolgáltatáskomponensek összekötésére alkalmas összekötőkkel alakíthatjuk ki. Ezek lesznek a szolgáltatások interfészei. Az interfészeket paraméterekkel specifikálhatjuk, amelyre a tevékenységparaméter-csomópontokat használhatjuk. Jelölésére a csatlakozólábas jelölést alkalmazzák. A csatlakozóláb tulajdonképpen egy objektumfolyam-csomópont, amely egy speciális eseményhez kapcsolódik. Megkülönböztetik a bejövő – az adatfolyamok befelé folynak – és a kimenő – az adatfolyamok kifelé folynak – lábakat. Az irány nyíl segítségével is feltüntethető. Jelölés segítségével az is megadható, hogy a lábak adatfolyamokat, adathalmazokat vagy kivételeket szolgáltatnak vagy fogadnak.



5.9. ábra: Csatlakozólábak típusai: adatfolyam feldolgozás (a,b), kivételkezelés (c), be/kimenő lábak (d), adathalmazok (e)

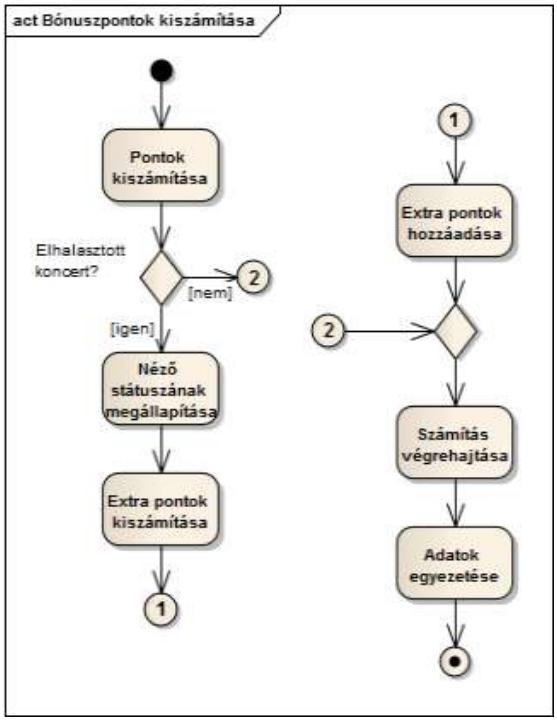
Egy tevékenység több paramétert is fogadhat, illetve nyújthat. Ilyen esetben a modellezés egyszerűsítésére alkalmazható a paraméterhalmaz konstrukció. A paraméterhalmaz elemei minden esetben rendelkezésre kell álljanak, de egyszerre csak a be- vagy a kimenő paraméterek halmaza kerül feldolgozásra, illetve előállításra.

Algoritmikus lefutások kezelése

A tevékenységdiagramok egyszerűbb jelöléseivel megismerkedtünk az előző fejezetekben. A tevékenységek folyamán azonban van hogy olyan dolgokat is le kell kezelni, amely nem a természetes lefutásnak megfelelő, például megszakítások keletkeznek a folyamatban, többször lefut egy tevékenység vagy tevékenységsorozat egy feltételnek megfelelően, vagy egyszerre több adatot is fel kell dolgoznia egy tevékenységnek. Az UML-ben ennek modellezésére is lehetőség van.

Ugrások

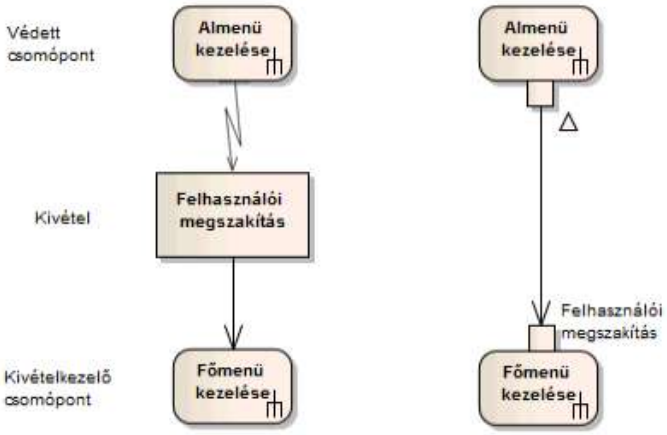
Előfordul, hogy egy tevékenységfolyam túl hosszú, túl bonyolult lesz, nem lehet a folyamot egyszerűen kezelni. Ilyen esetben használhatóak az ugrópontok, amelyek azt szimbolizálják, hogy egy folyam hol folytatódik. Jelölésbeli összekötőről van tehát szó. Egy diagramban ez a jelölés a következőképen néz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



5.10. ábra: Ugrójelek alkalmazása átfedő tevékenységfolyamok esetén

Kivételek

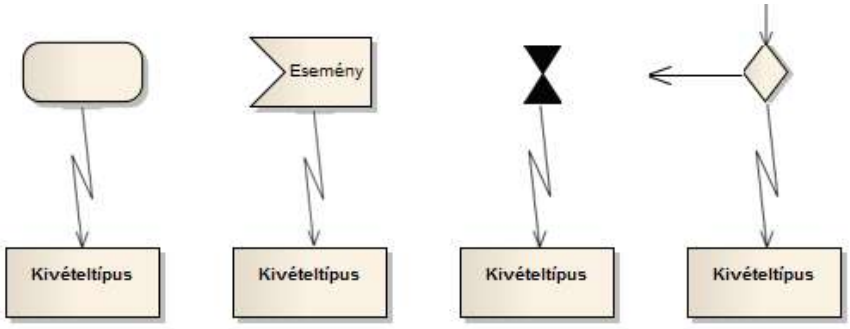
A kivételek kezelésére olyan esetben van szükség, ha pl. egy hiba hatására a tevékenységet meg kell szakítani, és a hiba kezelése a tevékenységen kívül folytatódik tovább. Ennek jelölése az 5.11-es ábrán látható módon történik.



5.11. 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. A kivétel négyféle módon váltódhat ki: közvetlen esemény, külső esemény hatására, időpont szerint vagy esetválasztással. Ezek jelölését szemlélteti az 5.12-es ábra.

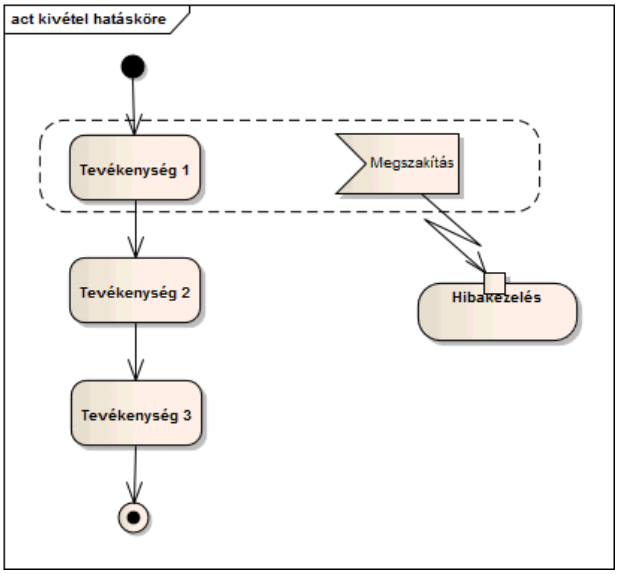


5.12. ábra: A kivételek kiváltásának módjai: közvetlen esemény, külső esemény, időpont hatására és esetválasztásként.

- Tevékenység: egy általánosan ismert tevékenység váltja ki az eseményt
- Külső esemény: esemény lép fel egy feldolgozás alatt lévő tartományban
- Időpont: speciális feldolgozás válik esedékessé egy adott időpontban
- Esetválasztás: egy feltétel alapján esetválasztás következik be és a választott ágon kivételkezelés történik

Megszakítási területek

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 5.13-as ábra.

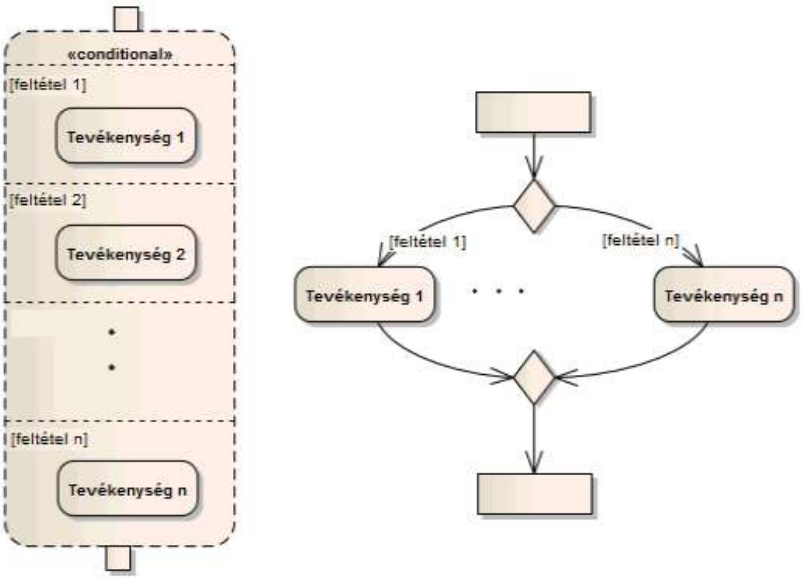


5.13. ábra: Kivétel kiváltása a megszakítási területen belül szakítja csak meg a vezérlőfolyamot.

Esetválasztó és ciklusszervező csomópontok

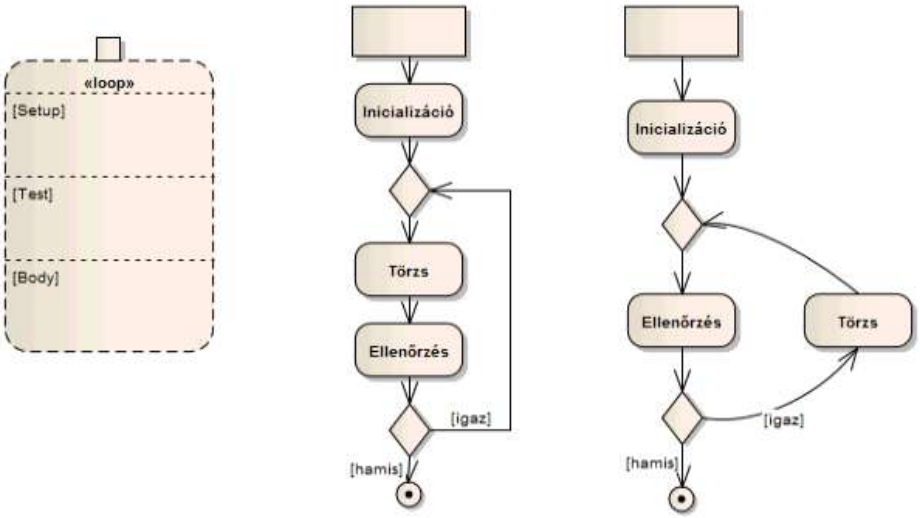
A strukturált csomópontok szerepe, hogy olyan lépéseket a vezérlőfolyamban, amelyek többször ugyanolyan módon futnának le, egyszerűsíteni lehessen. Ez nagyon hasonló a programozásban használt ciklusokhoz.

Esetválasztó csomópontokat olyan helyzetben használhatunk, ha egy feltétel hatására egy tevékenység folyik le, egy másik feltétel hatására egy másik tevékenység, és így tovább. A programozásban ismert case utasításnak felel meg a struktúra.



5.14. ábra: Esetválasztó csomópont

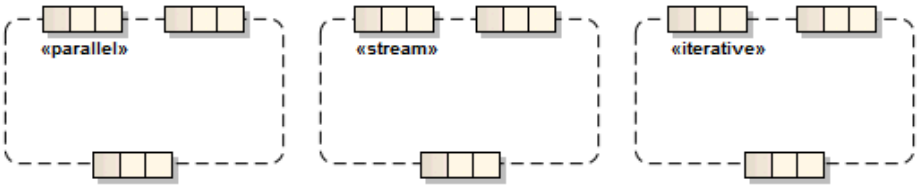
A ciklusszervező csomópontok az until- és a while ciklusoknak megfelelő struktúrát valósítják meg. Mindkettő azonos felépítésű: Inicializációból, Törzsből és Ellenőrzésből állnak. A ciklus akkor fejeződik be, ha az ellenőrzés negatív értéket ad.



5.15. ábra: Ciklusszervező csomópontok struktúrája

Kifejtési régiók

Az UML-ben is lehetséges az adathalmazok együttes kezelése. Az adathalmazok nem mások, mint azonos típusú adatelemekből álló adategyüttesek. Az adathalmazok feldolgozásának több lehetősége van, attól függően, hogy az adathalmazok hogyan érkeznek a feldolgozás pontjára. Ilyen lehetőségek a folyamként, párhuzamosan vagy szekvenciálisan érkező adatcsoportok. A feldolgozási módjukat az UML a stream, parallel, és az iterative kulcsszavakkal jelöli.



5.16. ábra: A kifejtési régiók három fajtájának jelölése

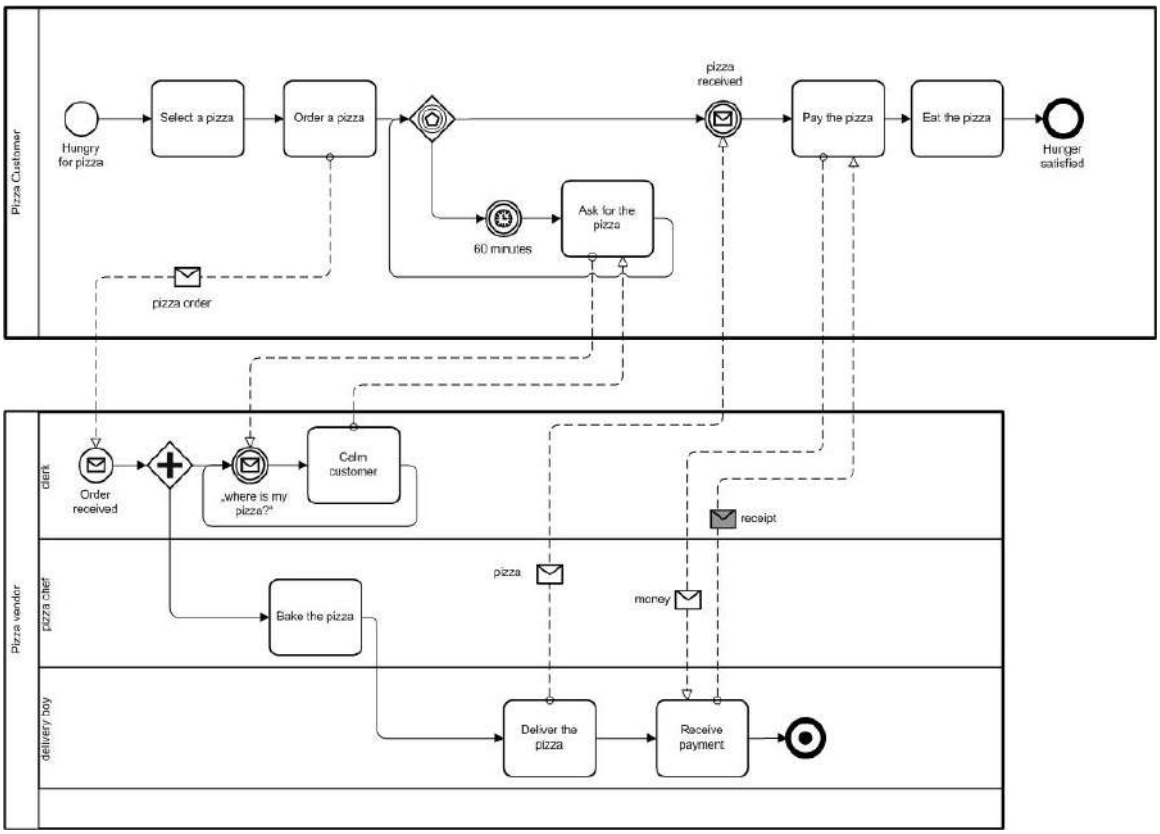
BPMN – az üzleti folyamatok modellezése

Érdekességképpen pár szót a tevékenységdiagramok specializációjáról:

A tevékenységdiagramok egy kifejezetten üzleti folyamatok számára továbbfejlesztett változata a BPMN[4] (Business Process Modeling Notation) jelölésrendszer diagramtípusa a BPD (Business Process Diagram). Ez a diagramtípus az UML-től függetlenül fejlődik, jelenleg a 2.0 verziónál tartanak. A jelölésrendszer alapja a tevékenységdiagramok jelölésrendszere. Az ábrán látható egy pizzaszállítási folyamat egyszerű üzleti folyamat diagramja.

A diagramok alapja valamilyen cselekmény illetve feladat, amit el kell végezni. Itt is megjelennek bizonyos események, mint pl. az üzenetküldés, amely hatással van a folyamatra. Megjelennek a különböző objektumtípusok, vagy például itt is azonosítani lehet az egyes résztvevőket úgynevezett Uszodák vagy Sávok megjelenítésével. Az alábbi ábrán - amely egy pizzarendelés folyamatát modellezi - a „pizza customer” mint megrendelő és a „pizza vendor” mint a pizzasütöde tevékenységei egy-egy uszodában (swimlane) helyezkednek el. A megjelenő vizuális elemek segítik az üzleti folyamatok tervezőit és modellezőit az értelmezésben. Kifejezetten olyan objektumok, jelölések jelennek meg, amelyek az üzleti folyamatok jellemzői.

[4] http://www.bpmn.org/



5.17. ábra: Pizza rendelés üzleti folyamat diagramja[5]

[5] http://www.omg.org/

6. A rendszer logikai struktúrájának modellezése: osztályok, osztálydiagramok

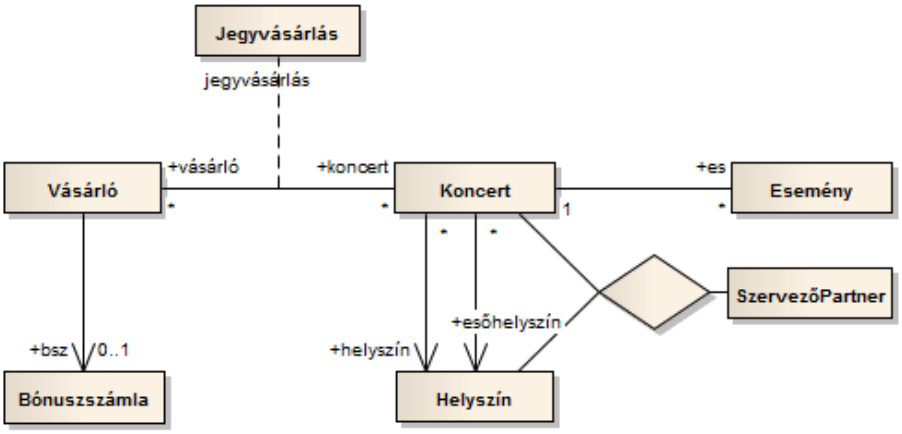
A rendszer folyamatainak feltérképezésével egyidejűleg meg kell tervezni a rendszer struktúráját is. Ennek lehetséges eszköze az osztálydiagram, amely a szoftvertervezési fázisok elemzési, tervezési és megvalósítási szakaszában is megjelenik, illetve a későbbiekben felhasználásra kerül. Az osztálydiagramok az objektumorientált modellek alapját képezik. Céljuk a valós világ objektumainak leképezése.

Az osztályoknak az UML-ben négyféle jelentést társítanak:

- Fogalom: fogalmak dokumentációjaként jelenik meg, mint például a szakterületi fogalmak
- Típus: ebben az esetben az osztály objektumai a típusok értékei lesznek, vagyis példányok
- Objektumhalmaz: a halmazhoz tartozó objektumok hasonlóan vannak deklarálva, ilyenkor az osztály egy csoportosítást jelképez
- Implementáció: a programozási nyelvekben megjelenő implementált osztályok

Elemzési osztálydiagram

Az elemzési osztálydiagramot a szakterület strukturális modellezésére használják. A szakterület folyamatainak modellezésére az előző fejezetben ismertetett tevékenységdiagramok valamint a következő fejezet objektum-életciklusai alkalmasak. A szakterület fogalmait és a kapcsolatokat a már megismert üzleti interjúk során ismerhetjük meg. Az alábbi elemzési osztálydiagramon ezek a fogalmak, mint egyszerű osztályok, attribútumok és metódusok nélkül jelennek meg.



6.1. ábra: A TF példarendszer szakterületének egyszerűsített osztálydiagramja

Az ábrán találkozunk a már megismert Vásárló, Jegyvásárlás, Koncert, Szervezőpartner, Helyszín, Bónuszszámla fogalmakkal, mint osztályokkal. Újként jelenik meg az Esemény osztály. A

Koncert osztály tulajdonképpen az azonos előadókat, számokat tartalmazó eseményeket fogja össze, amelyek csak a városban, helyszínben, a kezdési időpontban különböznek. Így pl. egy előadó több helyszínen is megtarthatja ugyanazt a tartalmú, pl. karácsonyi koncertjét.

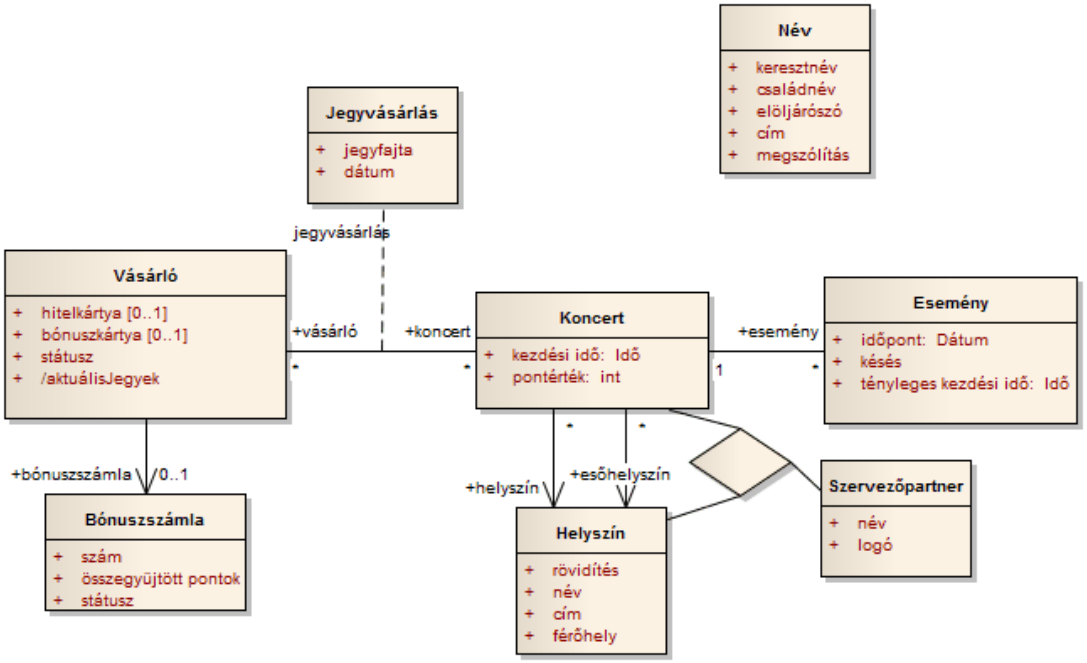
A fogalmak közötti kapcsolatokat az összekötők mutatják. Ezek az asszociációk a köztük lévő vonalak, amelyek mutathatnak irányultságot. Az irányultságot nyilakkal fejezhetjük ki. A rajzon ez jellemzően a helyszín és az esőhelyszín, amelyet a koncert osztály felől érünk el. Az asszociációkhoz multiplicitások, vagy más néven számosság kapcsolódhat. A Koncert és az Esemény közötti asszociációnál a koncerthez több esemény is tartozik, amelyet a *-al jelölt számosság mutat, míg egy esemény csak egy koncerthez tartozik, amelyet a koncert oldalán található 1-es jelöl. A Vásárló (Néző) és a Bónuszkártya közötti asszociációs kapcsolat számossága azt fejezi ki, hogy egy vásárlóhoz legalább 0 és legfeljebb 1 bónuszszámla tartozik. Az intervallumban bármilyen természetes szám állhat. Speciális osztály a Jegyvásárlás osztály, amely a vásárló és a koncert közötti asszociációhoz kapcsolódik. Az ilyen osztályokat asszociációs osztálynak nevezzük. Egy érdekes asszociációs kapcsolat látható a koncert, a helyszín és a szervezőiroda között. Ilyenkor a három osztály közötti egyenrangú asszociációról beszélünk.

Az osztálymodellben és a korábbi üzleti folyamat és használati eset modellekben ugyanazokkal a fogalmakkal találkozhattunk. Fontos, hogy a modellezés folyamán az azonos fogalmakat leképező modellező egységeket ugyanúgy nevezzük el. Ehhez segítségünkre lehet a fogalmi szótár megalkotása, amely a szakterületi fogalmakat és a leírásukat tartalmazza. A TF rendszer fogalmi szótárának egy részletét tartalmazza a 6.1-es táblázat.

6.1. táblázat: TF fogalmi szótárának egy részlete

Fogalom	**Szinonima**	**Magyarázat**
Jegyvásárlás	Jegyfoglalás	A vásárló az adott egy koncertre jegyet vásárol
Koncert	Koncertsorozat, eseménysorozat	A koncert az év egy részében, adott időpontokban, helyeken megrendezett esemény. Az esemény a Koncert egy adott időpontban, helyszínen megrendezett példánya
Esemény		Az esemény egy megrendezett koncert egy adott időpontban és helyszínen.
Név		A név a keresztnév és a vezetéknév illetve egyéb elöljárószók összessége
Vásárló	Néző, Ügyfél	A vásárló olyan személy, aki jegyet vásárolt a rendszerben

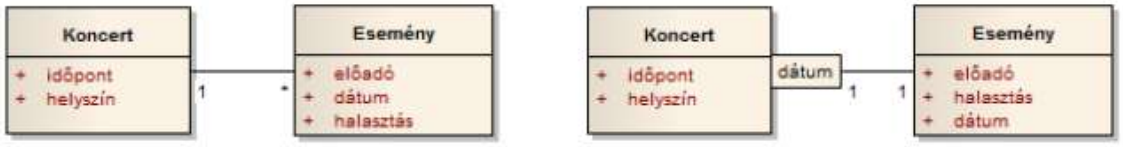
Az osztálydiagram osztályait a következő lépésben a rájuk jellemző adatokkal, attribútumokkal egészítjük ki. Az attribútumok az osztály neve alatti rekeszbe kerülnek. Az attribútumokat típussal is el lehet látni. Ez lehet egy alaptípus, mint például a string a keresztnév attribútumnál, vagy lehet másik osztály, mint pl. a Vásárló.Név attribútumérték. Az attribútumok melletti számértékek jelölik az attribútumok számosságát (a vásárló hitel- vagy bónuszkártyája), ugyanúgy ahogyan az asszociációk végei. Az attribútumok előtti + jelek az attribútumok láthatóságára történő utalás. A szakterületi szint osztályainál ezt nem szükséges feltétlenül feltüntetni, a későbbiekben részletesen kifejtésre kerülnek.



6.2. ábra: Az osztályok attribútumai

Az asszociációk és az attribútumok felcserélhetők, mindkettő egy mutató értéke lesz. A minősítők alkalmazása az asszociált objektumok hozzáférését könnyítheti meg, amivel az asszociáció számosságát lehet lecsökkenteni rendszerint 1-re, ezzel egyértelművé téve az elérést.

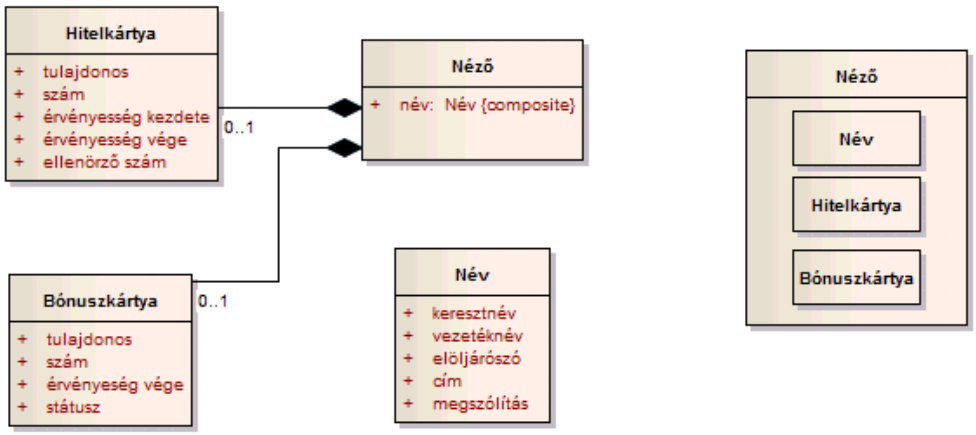
6.3. ábra: Az asszociációk és attribútumok egyenértékűsége



6.4. ábra: A minősítőkkel a számosság csökkenthető

Kompozíciós kapcsolat

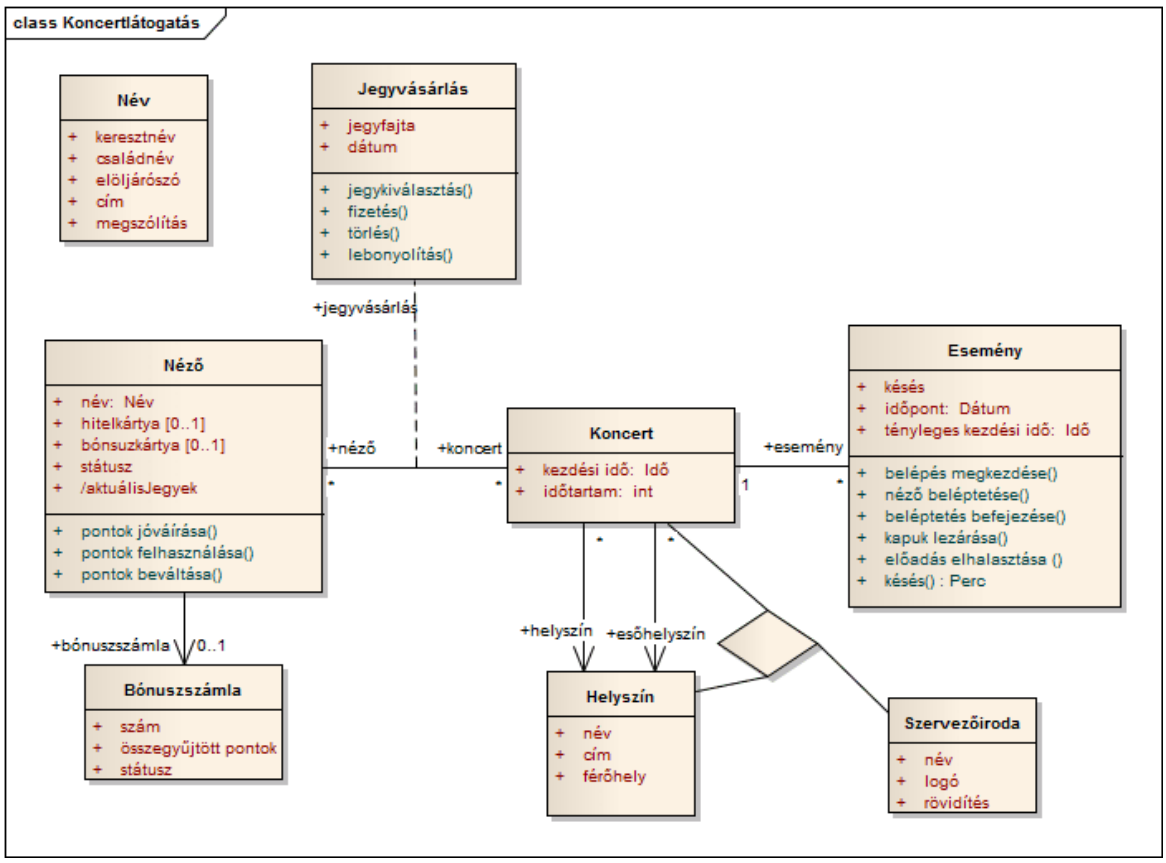
A kompozíciós kapcsolat az osztályok között rész-egész kapcsolatot jelöl, tehát egy objektum egy másik objektum részét képezi. Más objektum a részobjektumot nem birtokolhatja, és az objektum törlése esetén a részének is törlődnie kell. Egy rész akkor törlődhet, ha az objektum törlődik, vagy ha a kapcsolatuk számossága megengedi, például $[0\ldots1]$. A kompozíciót az asszociáció végénél elhelyezett fekete rombusszal jelöljük. A 6.5-ös ábrán kompozíciós kapcsolat van jelölve a Vásárló és a Hitel- valamint Bónuszkártyája között. Másik jelölési mód a $\{\text{composite}\}$ más néven összetett objektumként történő megjelenítés, illetve a beágyazás.



6.5. ábra: Kompozíciós kapcsolat jelölései

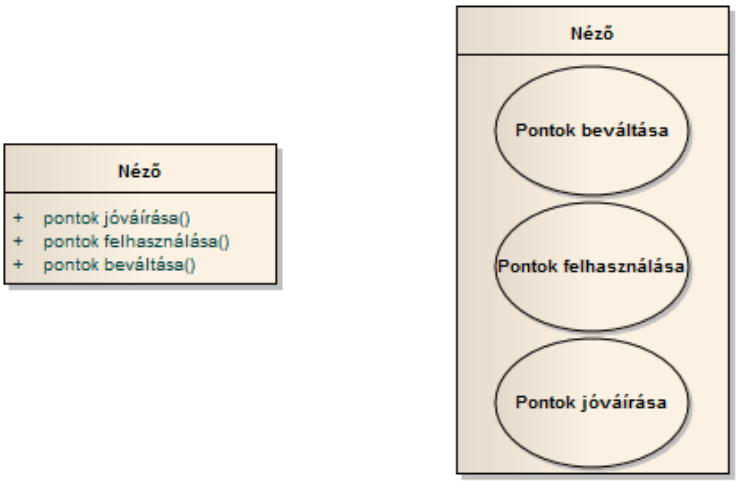
Metódusok

A metódusok a használati esetekhez hasonlóan a viselkedés leírására használhatóak, de amíg a használati esetek a rendszer viselkedésének, funkcionalitásának leírására alkalmasak, addig a metódusok az objektumok szintjén írják le a viselkedést. A metódusok megjelenítéséhez egy újabb rekeszt illesztünk az attribútumrekesz alá. Ezeken kívül több rekesz is csatlakoztatható az osztályokhoz, például tevékenységdiagramoknak vagy használati eseteknek is készíthetünk rekeszeket. A használati esetek elnevezése jól használható a metódusok elnevezéséhez.



6.6. ábra: Osztálydiagram metódusokkal kiegészítve

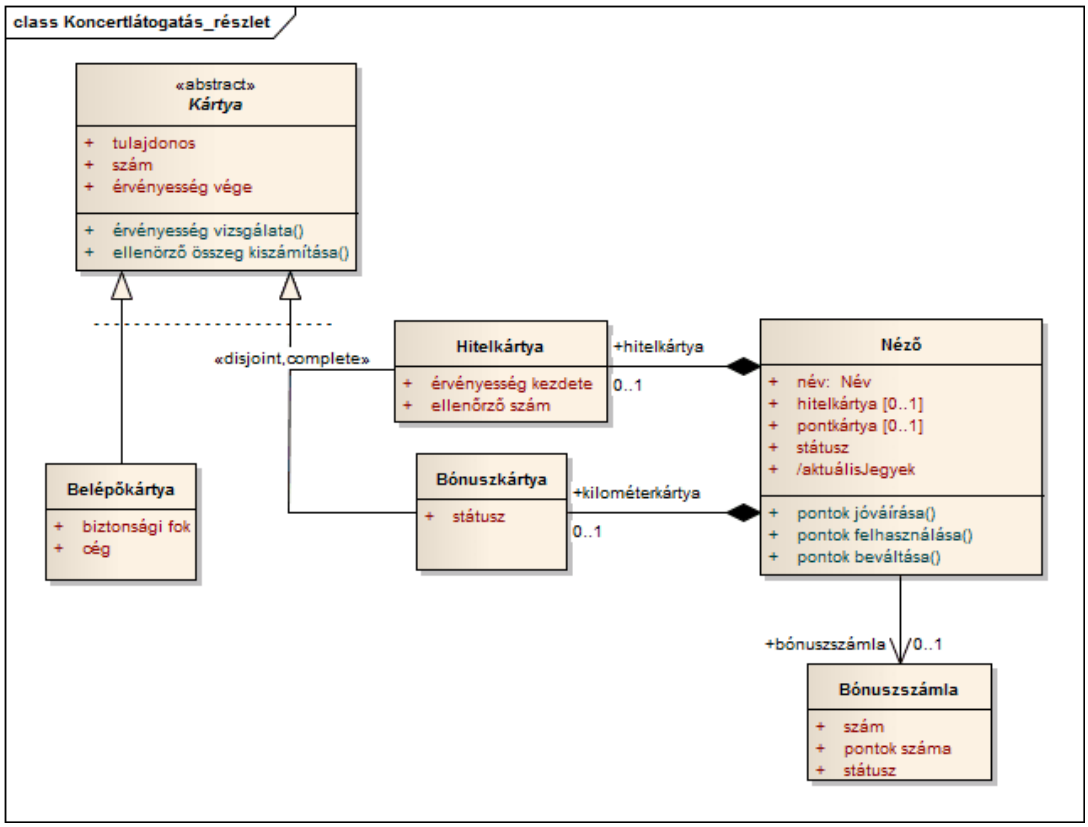
A 6.6-os ábra osztálydiagramján láthatóak a Nézőhöz, illetve az Esemény osztályhoz kapcsolt metódusok.



6.7. ábra: Használati esetek és metódusok elnevezései

Öröklődés

Az objektumorientáltság egyik fontos fogalma az öröklődés. Ezt az UML általánosításként kezeli és fehér hegyű nyílként jelöli. A nyíl a speciálistól az általános felé mutat.

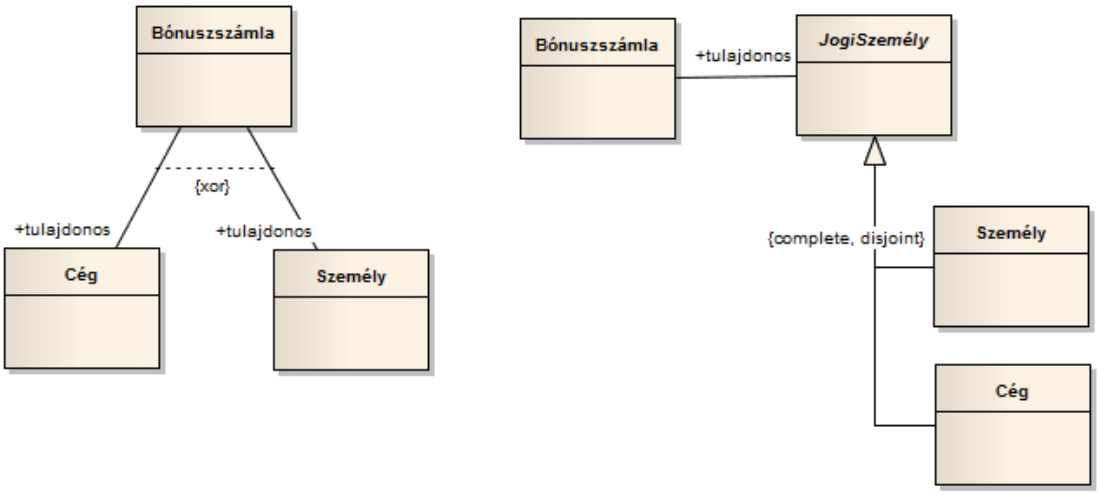


6.8. ábra: Általánosítás használata az UML osztálydiagramján

A Belépőkártya, a Hitelkártya és a Bónuszkártya a Kártya osztály speciális esetei. A Kártya osztály dőlt betűs jelölés azt jelenti, hogy egy absztrakt osztályról van szó. Az absztrakt osztály olyan osztály, amely nem példányosítható. Az általánosítási kapcsolatok általánosításhalmazokba foghatóak meghatározott szempontok szerint. Az 6.8. ábraa Hitelkártya és a Bónuszkártya általánosítási kapcsolata általánosításhalmazba van foglalva, amelyet a közös nyíl is jelöl. A nyílon lévő disjoint és complete tulajdonságok jelentése a következő:

- disjoint: az általánosításhalmazon belüli általánosítások diszjunktak
- complete: nincsenek további általánosítások az általánosításhalmazon belül. Az ősosztály minden példánya az általánosításhalmazban lévő alosztály egy példánya kell legyen

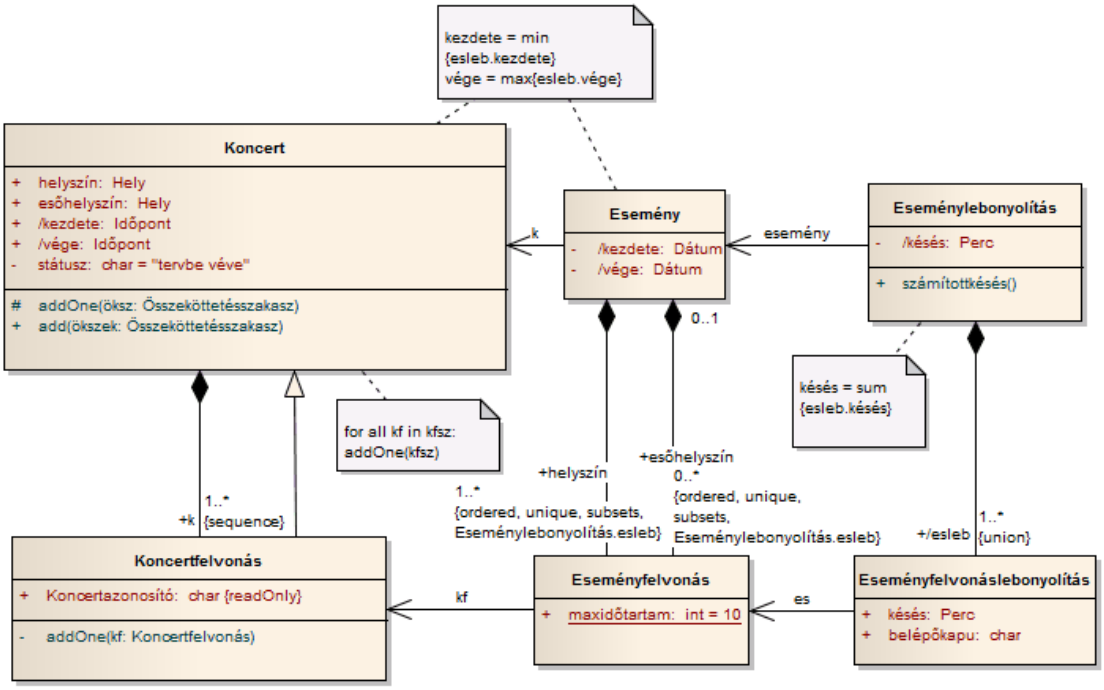
Az öröklési kapcsolat nem csak az osztályokra, hanem az asszociációkra is alkalmazható. Az asszociációkra bizonyos megkötéseket is tehetünk. Az alábbi ábrán látható xor megszorítás azt jelenti, hogy a Bónuszkártya vagy egy Céghez vagy egy Személyhez tartozhat. Egy másik lehetséges megoldás erre egy újabb absztrakt osztály bevezetése (6.9. ábra).



6.9. ábra: Megszorítások alkalmazása asszociációk között. Xor megszorítás

Tervezési osztálydiagram

Az elemzési osztálydiagramok után a tervezési osztálydiagramok már nem a megvalósítandó rendszerről, hanem a megvalósítás módjáról szólnak. Még nem a megvalósítás szintjén járunk, de közelebb jutunk hozzá. A technikai szempontok is előtérbe kerülnek. Itt is megjelennek az attribútumok és a metódusok, de már egy mélyebb szinten, jobban finomodik a modell. A TicketFirst rendszer egy részének tervezési osztálydiagramja a következőképpen írható le (6.10. ábra):



6.10. ábra: A TF rendszer tervezési osztálymodelljének egy részlete

Attribútumok

A tervezési osztálydiagram attribútumai többféle tulajdonsággal vértezhetőek fel. Ezek egyike a láthatóság. A osztályok ezen a szinten mint egymásba ágyazott névterek jelennek meg. A láthatóság a névtereken belül és köztük értelmezett. A láthatóság értékei következőek:

- public + : az attribútum az egész névtérben látható
- protected # :az attribútum olyan névtereken látható, ahonnan általánosítható kapcsolat áll fenn az adott névtérhez
- package ~ : az attribútum csak ugyanazon csomag elemei számára látható
- private - :az attribútum csak a közvetlenül tartalmazó névtérben látható

Az attribútum előtti törtjel (/) azt jelenti, hogy az attribútum máshonnan van származtatva. Ez lehetséges öröklődéssel, vagy más attribútumból kiszámított értékkel.

Olyan attribútumokat is meghatározhatunk, amelyek csak az osztályhoz és nem a származtatott attribútumhoz tartoznak. Ezt aláhúzással jelöljük.

Az attribútumok még további tulajdonságokkal is elláthatóak. Ugyanez igaz az asszociációkra is. A specifikált tulajdonságokkal összetett megszorításokat lehet meghatározni, amelyek a rendszer futása idején értékelődnek ki.

- readOnly: az attribútum a kezdő értékadás után nem módosítható. A származtatott attribútumok gyakran kapják meg ezt a tulajdonságot
- composite: összetett attribútumot jelöl
- redefines Attr: az ősosztály attribútuma az alosztályban felülírásra kerül

- subsets Attr: nagyobb számossággal rendelkező attribútum esetén ezzel a tulajdonsággal adható meg, hogy az attribútum elemi az Attr attribútum részhalmazából valók
- union: ha egy névtérben egy attribútum a subsets Attr tulajdonsággal van ellátva, akkor az Attr a union tulajdonsággal deklarálható, amely azt jelenti, hogy Attr a subset-el ellátott részhalmazainak unióját alkotja
- unique: az attribútum értéke csak egyszer fordul elő. Ellentéte a nonunique, az alapértelmezés a unique
- ordered: egynél nagyobb számosság esetén értelmes deklarálni mivel azt jelenti, hogy az elemek rendezettek. Ellentéte az unordered, alapértelmezés az ordered
- sequence: egynél nagyobb számosság esetén értelmes a deklaráció. Az attribútum elemei listában tárolódnak el
- bag: az attribútum elemei multihalmazként vannak tárolva

A tulajdonságértékek kiírása opcionális. Az attribútumdeklarációk megadásának pontos szintaktikája a következő:

Attribútum ::= [Láthatóság] [/] Név [:Típus] [Érték]

Láthatóság ::= +|#|~|-

Érték ::= [[Számosság]] [=Érték] [{Tulajdonságok}]

Számosság ::= Korlát […Korlát|*]

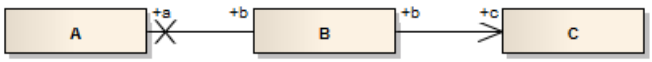
Korlát ::= 1|1|…

Tulajdonságok ::= [readOnly|unrestricted] [composite] [unique|nonunique] [ordered|unordered] [bag] [seq|sequence] [union] (subsets Attr)* (redefines Attr)*

Kapcsolatok

Az asszociációs kapcsolatok között vannak irányított és irányítatlan asszociációk. Ahogyan már említésre került, az asszociációk egyenértékűek az attribútumokkal. A kapcsolatok végén lévő feliratok így azonosak az attribútumokkal. A sorrendet és az elrendezést a szabvány nem határozza meg.

Az osztályok közötti navigációnál nemcsak az irány meghatározására van lehetőség, hanem arra is, hogy megtiltsuk egyik osztályból a másikba történő navigációt. A navigáció irányát az asszociáción nyíllal jelöljük, a navigáció tiltását pedig egy kereszt jelöli az asszociációvégen. A 6.11-es ábrán látható, hogy a B osztály felől a C osztály elérhető, az A osztály viszont nem. Így a B.a nem érvényes kifejezés. Ha nem szerepel nyíl vagy kereszt az asszociáción, az azt jelenti, hogy a navigálhatóság nincs korlátozva.



6.11. ábra: Navigációs útvonalak

Műveletek

Az attribútumokhoz hasonlóan a műveletek is felruházhatók bizonyos tulajdonságokkal. A láthatóságra, számosságra, visszatérési értékre vonatkozó és egyéb tulajdonságaik az attribútumoknál ismertetett tulajdonságokkal egyeznek meg. Ezeken kívül itt egy paraméterlistát is meg lehet határozni, amelyek mindegyikéhez meg lehet adni név, irány, típus, számosság és alapértelmezett értékeket. A deklarációk szintaktikája a következő:

Művelet ::= [Láthatóság] Név ([Paraméterlista]) [:Típus] [{Tulajdonságok}]

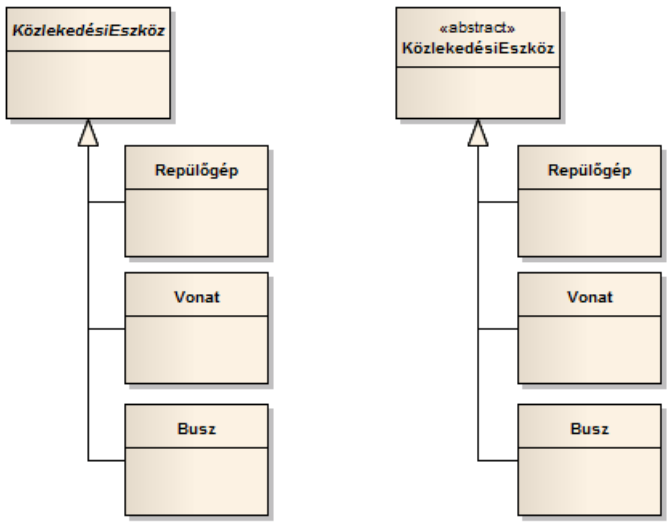
Paraméterlista ::= Paraméter (, Paraméter)*

Paraméter ::= [Irány] Név: Típus Érték

Irány ::= in |out|inout|return

Absztrakt osztályok

Az olyan osztályokat, amelyeknek alosztályai lehetnek, de példányai nem, absztrakt osztályoknak nevezzük. Az absztrakt osztály ábrázolása a következő módokon lehetséges (6.12. ábra):



6.12. ábra: Absztrakt osztályok ábrázolása

Az absztrakt osztályok általában azokat a viselkedésre vonatkozó nézőpontokat tartalmazzák, amelyek az alosztályokban is megjelennek.

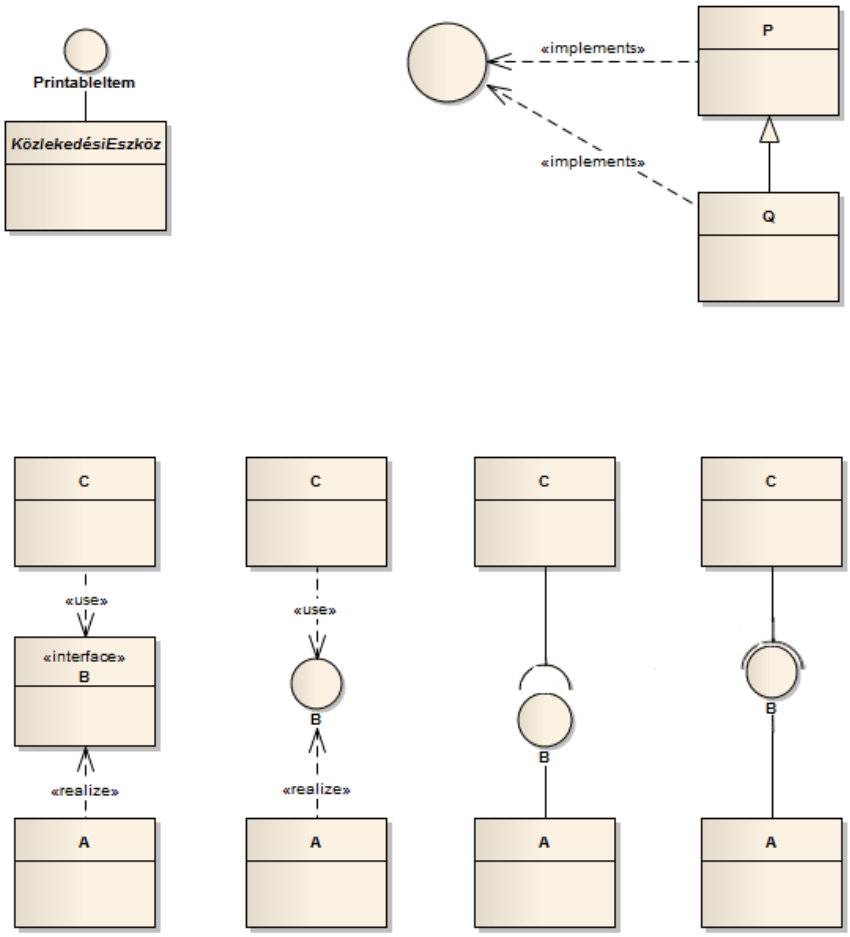
Aktív osztályok

Vannak olyan osztályok, amelyek önmaguktól aktiválódnak, nem szükséges más külső esemény az elindításukhoz. Ilyenek például az operációs rendszer folyamatai. Ábrázolásmódjuk a következő lehet:

6.13. ábra: Aktív osztályok ábrázolásának lehetőségei

Interfészek

Az interfészek az osztályok által várt vagy nyújtott funkcionalitásokat, műveleteket testesítik meg. Metódusokat nem tartalmaznak, csak specifikációkat. Jelölésük a következőképen valósul meg (6.14. ábra):



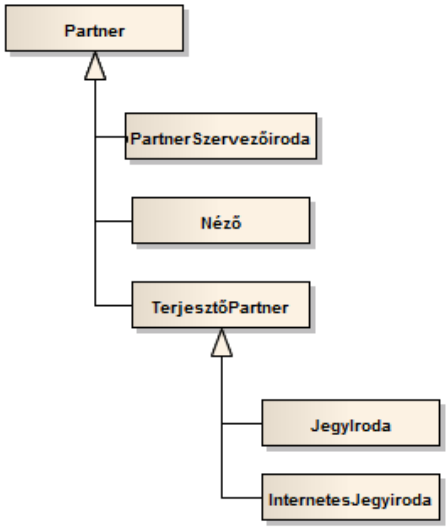
6.14. ábra: Interfészek ábrázolása

Az A a B interfészt nyújtja (realize), a C a B interfészt követeli meg (use). Tehát a nyújtott interfészt a realize, a megkövetelt interfészt a use kulcsszavakkal jelezzük. Ha egy osztály implementál egy interfészt, akkor az interfész felé mutató szaggatott nyíl implements feliratot kap az osztályból.

Speciális osztálydiagramok

Taxonómia

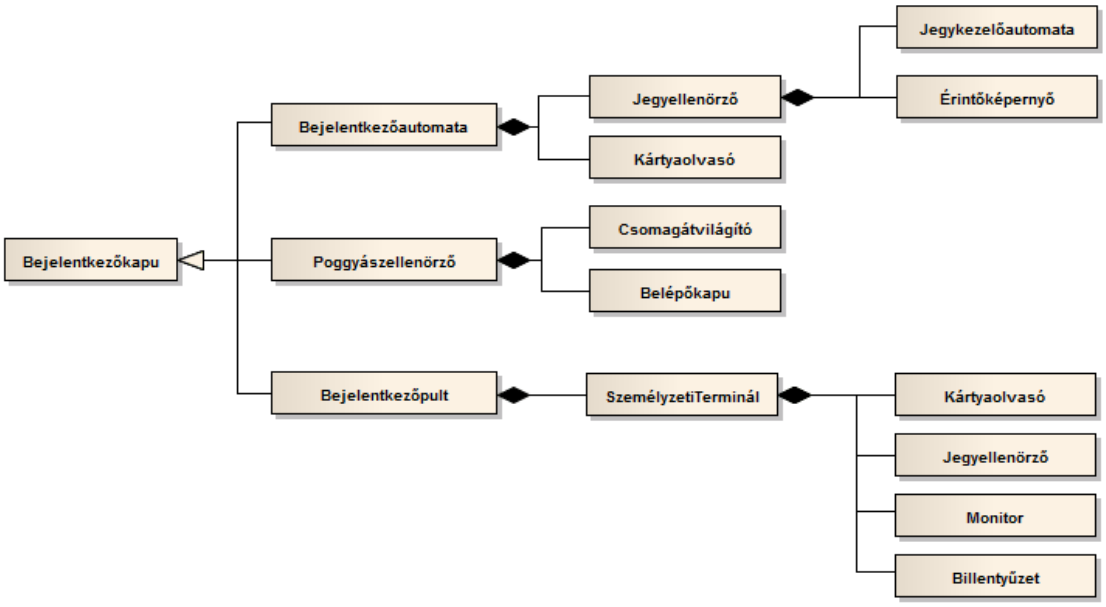
Az osztálydiagramok speciális esetei, amely az osztályok közötti öröklődés hierarchikus struktúráját mutatja meg. Csak az osztályok közötti kapcsolatok szerepelnek rajta, attribútumok és metódusok nem. Egy taxonómiának az összes osztály és öröklődési kapcsolatot tartalmazni kell, teljesnek kell lennie. A szoftverfejlesztés összes fázisában jól használható, az osztályokat nézhetjük az elemzési, szakterületi szinten, de az implementációs osztályok taxonómiáját is el lehet készíteni.



6.15. ábra: A TF rendszer partnereinek taxonómiája

Kompozíció hierarchia

Az osztályok közötti kapcsolatok megjelenítésének egy másik lehetséges módja a kompozíció-hierarchia. Ennél a típusnál a nevéből is adódóan az osztályok kompozíciós kapcsolataira építünk. A hierarchia felépítése nem tűnik nehéznek, de ahhoz, hogy logikailag megfelelően építsük fel, a rendszer teljes átgondolása, és a modell többszöri átrajzolása lehet szükséges. Ha megfelelő módon építjük fel a hierarchiát, az nagyban segíteni fogja modellünk áttekinthetőségét. A 6.16-os ábrán látható a Beléptetés osztályainak kompozíció hierarchiája.

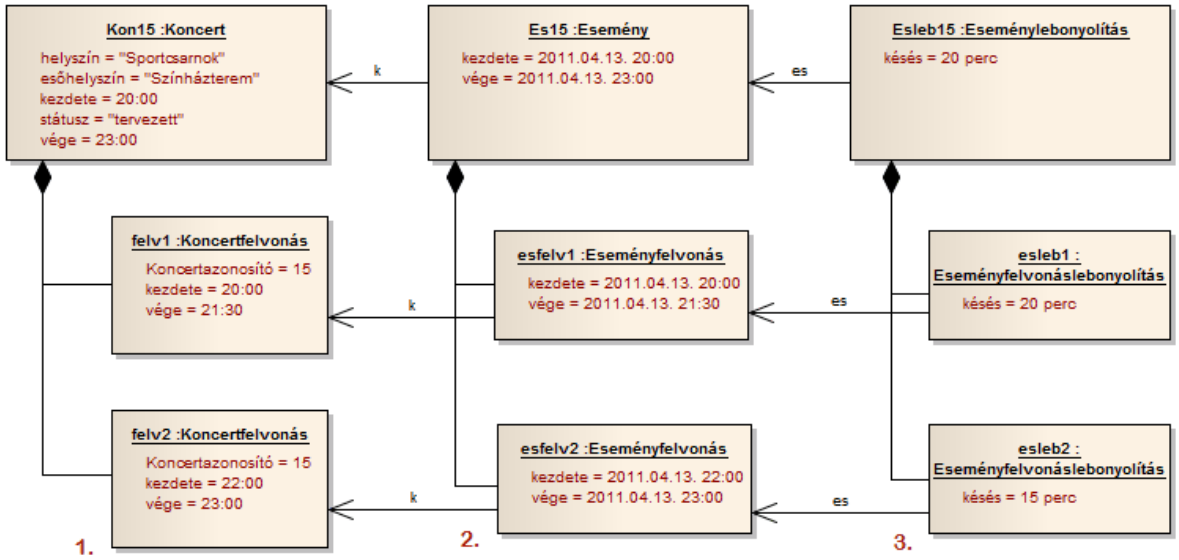


6.16. ábra: A Beléptetés osztályainak kompozíció hierarchiája

7. Az osztálydiagramok megjelenése a szoftverfejlesztésben

Objektumdiagramok

Az osztálydiagramok a rendszer strukturális modellezésének alapkövei. Meghatározhatóak a rendszer lehetséges kivitelezéseinek halmazai. Egyes esetekben szükség lehet arra, hogy egy ilyen kivitelezési állapotot modellezzenek, erre pedig az UML objektumdiagramjai használhatók. Az objektumdiagramokban az osztályok példányai jelennek meg. Ezeket a nevük aláhúzásával jelöljük. A diagramon az osztályok példányainak több állapota is megjelenhet. Az ábrán a koncert egyes osztályaira jellemző rendszerállapotok átható. Az első oszlopban a jegykeresés után, a második oszlopban a jegyvásárlás után, a harmadik oszlopban pedig a koncert befejezése utáni állapot látható. A példányok az egyes rendszerállapotokban példányosulnak.



7.1. ábra: Rendszerállapotok szemléltetése objektumdiagrammal

Szerepkörök

Egy objektum egyszerre több szerepet is elláthat. Más objektumokkal összeköttetésben állhat, és ezekkel együttműködhet. A működés során többféle szerepe lehet az objektumnak. Például egyszer szolgáltatást nyújthat, máskor pedig szolgáltatást kér egy másik objektumtól. Ezeknek a szerepeknek a leírására szolgál az együttműködéseken belüli szerepkör fogalma. A szerepkör nem egy külön osztály vagy objektum, hanem egy objektumra vonatkozó viselkedés egy adott együttműködésen belül. Az objektum összes működését egy ilyen helyzetben leszűkítjük az adott együttműködés számára szükséges jellemzőkre. Nagyon hasonlít a működés az interfészek működésekre, de a szerepkörök attribútumokkal is rendelkeznek. Az említett leszűkítésre sem az

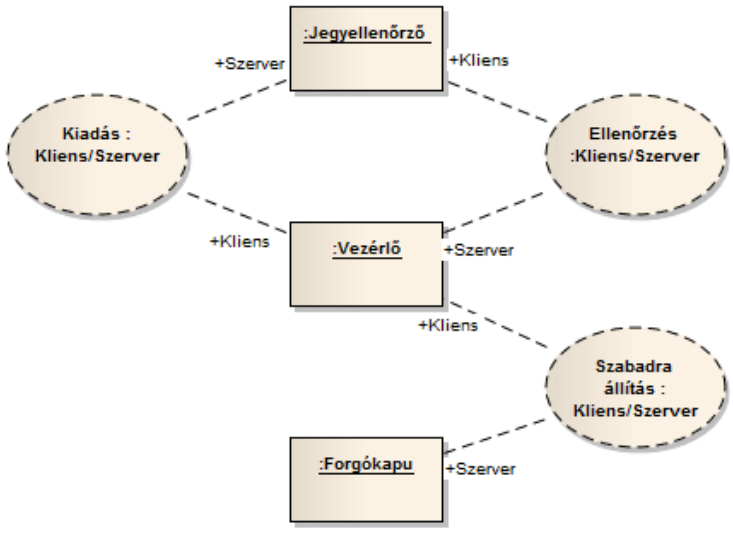
osztálydiagramok, sem az objektumdiagramok nem alkalmasak, mindkettőnél az osztály illetve objektum összes szerepköre modellezve van. A szerepkörök neve a következőképen adható meg:

Szerepkörnév ::= ([név[/szerepkör]]|/szerepkör) [:típus(,típus)]

A szerepkörök a kliens-szerver együttműködéssel jól bemutathatóak. A koncertre történő bejelentkezés jól példázza az objektumok több szerepkörben történő részvételét a folyamatban. A beléptetés során a következő történik:

1. A rendszerbe betöltik a koncertre érvényes jegyek sorszámait. A jegyellenőrző automatát és a forgókaput ellenőrzik és inicializálják.
2. Ha egy jegyet helyeznek a jegyellenőrzőbe, akkor leolvassa a sorszámot és továbbítja a vezérlőnek.
3. A vezérlő megvizsgálja, hogy az adott sorszámú jegy megvételre került-e és az aktuális koncertre érvényes-e. Ha igen, akkor érvényesíti a jegyet és visszaadja a Néző nek. Eltárolja a jegy sorszámát és a forgókaput szabadra állítja.
4. A forgókapu az elfordulás után lezár.
5. Ha valami probléma áll fenn az ellenőrzés során, pl. a jegy nem olvasható, más gond van a jeggyel, nem erre a koncertre érvényes, akkor ezt jelzi a rendszer a személyzetnek.

A jegyellenőrző automata részeinek szerepkörei a folyamatban a következőképpen modellezhetőek:

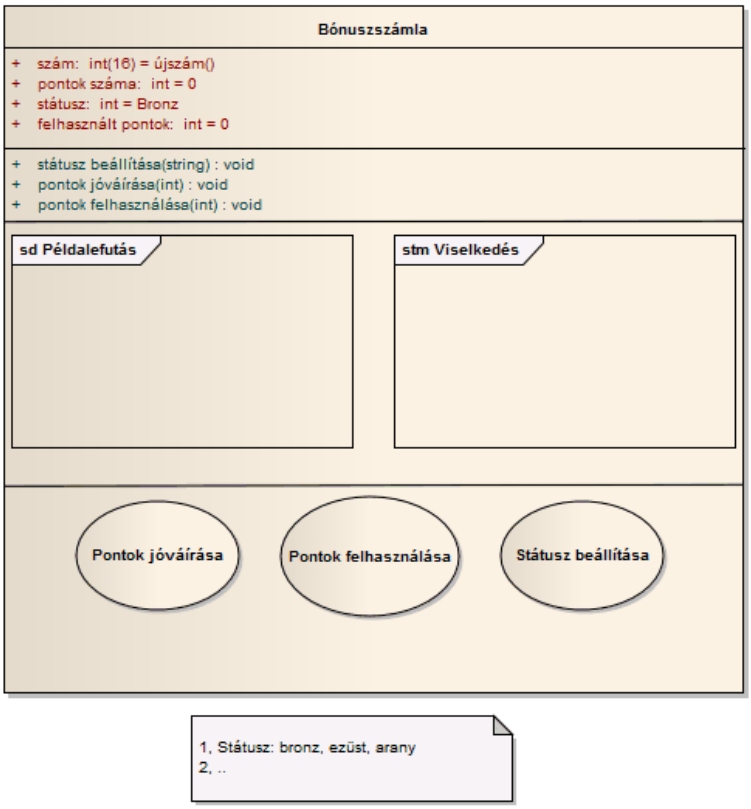


7.2. ábra: A jegyellenőrző automata alrendszereinek szerepkörei

Ez a leírás állapotautomatával is modellezhető, ehhez azonban több információra lesz szükség a rendszer működésével kapcsolatban.

Osztályleltár

Az osztályleltár megalkotása fontos a rendszer dokumentációjának elkészítéséhez. A rendszer összes osztályának leírását tartalmazza az osztályok minden jellemzőjével együtt. Ezt táblázatos formában vagy teljesen részletezett osztályként is meg lehet tenni. Az osztályleltárt leginkább a rendszer tervezési fázisában szokták elkészíteni.



7.3. ábra: A bónuszszámla osztályleltára

Ahogyan a 7.3-as ábrán látható, a leltárban megjelennek az osztály attribútumai, a metódusok, a tevékenységek leírását szolgáló diagramok, illetve a használati esetek. Mindegyik számára egy külön rekesszel egészítődik ki az osztály. Lehetőség van megszorítások leírására is, ez megjegyzésdobozban jelenik meg.

Az osztályleltár táblázatos formában a következőképpen néz ki (7.1. táblázat):

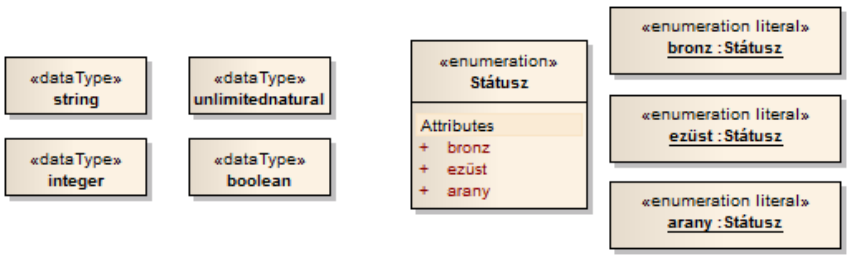
7.1. táblázat Bónuszszámla osztályleltárának táblázata

Bónuszszámla			
Attribútum	**Típus**	**kezdőérték**	**Magyarázat**
szám	int	automatikus	generálódik
bónuszpontok száma	int	0	lejárati dátummal
státusz	string	0	bronz
felhasznált bónuszpontok	int	0	utolsó beváltás dátuma
Metódus	**Típus**	**Paraméter**	**Magyarázat**
státusz beállítása	void	újStátusz:string	
bónuszpontok jóváírása	void	pontok:int vásárlástípus:char	koncertjegy vagy egyéb vásárlás
bónuszpontok felhasználása	void	pontok:int felhasználástípus:char	

Megvalósítási osztálydiagram

Az elemzésen és a tervezésen átlépve elérünk a megvalósítás fázisához. A megvalósítás fázisa már teljesen a technológiai részletekre koncentrál, amely az elemzés és a tervezés során azonosított követelményekre, funkciókra és struktúrára épül. A megvalósítási osztálydiagramban többféle attribútum- illetve metódustípussal találkozhatunk, a modellezés eszköztára bővebb.

Az alap adattípusokon kívül használhatunk felsorolásos típusokat is. Az alaptípusok strukturálatlan adattípusok, külsőleg definiált értékkel és műveletekkel. Az UML-ben a boolean, az integer, az unlimitednatural és a string alaptípusok használhatók, de csak az elemzési és a tervezési osztálydiagramokon jelennek meg. A megvalósítási diagramokon, a megvalósítási nyelv típusai jelennek meg, legyen ez Java, vagy C++ vagy más objektumorientált programozási nyelv. A felsorolásos típusok olyan előre definiált típusok, amelyek egy véges halmazból veszik az értékeiket. A felsorolásos típust az Enumeration-nel specifikáljuk.

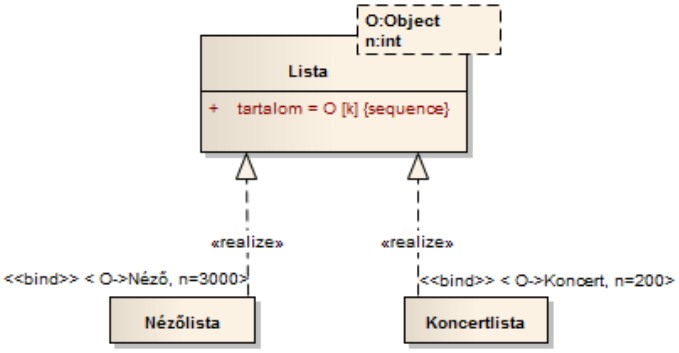


7.4. ábra: Az UML adattípusai

A jobb oldalon a felsorolásos típus két egyenértékű ábrázolása látható, jobb oldalon az alaptípusok.

Sablonosztályok

AZ UML-ben a sablonosztályok a programozási nyelvekben megismerhető parametrikus polimorfizmus egy formájának modellezésére alkalmasak.



7.5. ábra: A parametrikus polimorfizmus az UML-ben

A Lista osztály egy sablon, amelyből példányokat lehet készíteni. Ahhoz viszont, hogy ez megtörténhessen, a szabadon hagyott típusparamétereket meg kell határozni. Így létrehozható belőle a Koncertlista és a Nézőlista. Mindkét listához meghatározásra kerülnek a paraméterek.

A megvalósítási osztálydiagram profilja

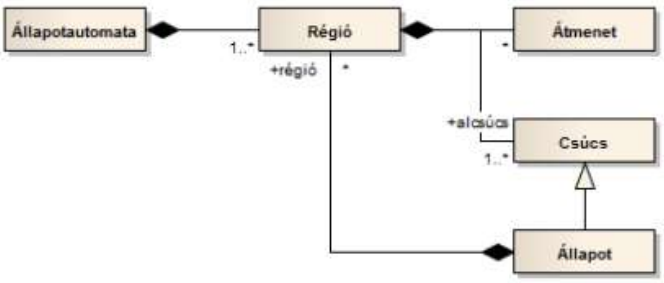
Az implementáció során éppen alkalmazandó megvalósítási nyelvhez egy profilt szoktak létrehozni. A profilokban néhány kifejezőeszköz, illetve megszorítások is rendelkezésre állnak. Ha egy modell ennek a profilnak megfelel, akkor osztályai közvetlenül leképezhetőek egy implementációs nyelv osztályaira. Innentől fogva a munka modellvezérelt módon végezhető, és a modellben elvégzett módosítások egy fordító segítségével átvihetőek az implementációba. Nagyobb rendszerek újratervezésénél is nagy segítségünkre lehet.

Az implementációs nyelv meghatározása a projekttől függ. Bizonyos profilok letölthetők az OMG lapjáról, de ezeken projektspecifikus változtatásokat kell elvégezni.

Az UML megvalósítási osztálydiagramjai Java- vagy C++ programként értelmezhetőek , az elemzési osztálydiagram pedig mint logikai feladatmodell tekinthető.

8. Objektumállapotok modellezése: állapotautomaták

Az állapotautomaták annyiban hasonlítanak a már tárgyalt tevékenységdiagramokhoz, hogy ezekkel is folyamatok lépéseit írjuk le, de jelen esetben nem a tevékenységekre, hanem az állapotokra és az állapotátmenetre koncentrálunk. Az állapotautomaták régiókat tartalmaznak, amelyek csúcsokból és átmenetekből állnak. A csúcsok állapotokat jelölnek, amelyek régiókra tagolódhatnak.

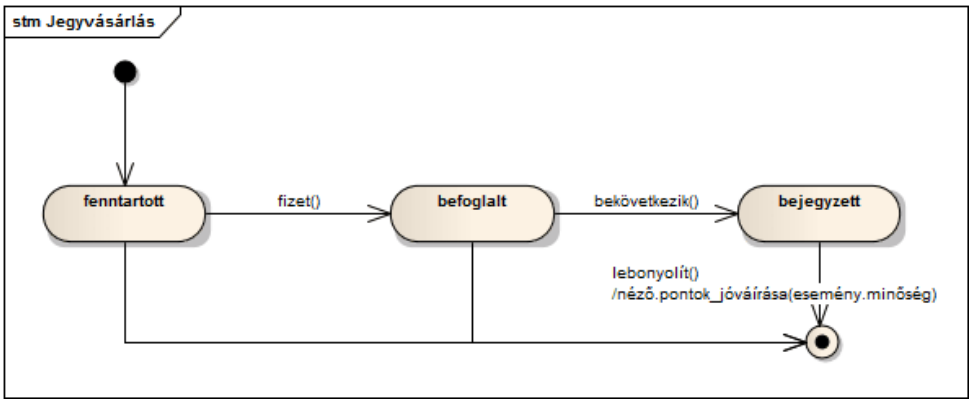


8.1. ábra: Az állapotautomaták struktúrája osztálydiagrammal modellezve

Az állapotautomaták tehát hierarchikus felépítésűek. Az állapotok és alállapotok fáját állapot-konfigurációnak hívják.

Objektum-életciklus

Az objektumok viselkedése leírásának egyik módja az életciklusuk leírása állapotautomata segítségével. Ehhez az osztály elérhető műveletei használhatóak. A 8.2. ábrán a Jegyvásárlás osztály objektum-életciklusa látható.



8.2. ábra: A Jegyvásárlás osztály példányainak életciklusa

Az ábra alapján a Jegyvásárlás három állapotban lehet, ez a fenntartott, a befoglalt és a bejegyzett állapotok. A fenntartott állapot az, amikor a vásárló lefoglalja a kereset koncertjegyet, befoglalttá akkor válik, amikor a jegy kifizetésre került, és akkor válik bejegyzetté, amikor a

koncert lezajlik. A kezdőállapotot a fekete pont, a végállapot pedig a kettős kör jelzi, amelynek telített a belseje. Az állapotok közötti átmeneteket állapotváltozások jelenítik meg, amelyek nyilakkal vannak jelölve. A nyilakon kiváltó ok, feltétel és hatás szerepelhet megjegyzésként. Ha az átmenetnek nincs kiváltó oka, akkor teljesülő átmenetről beszélünk. Ilyenkor az előző állapot befejezésével jutunk át a következő állapotba. Kiváltó ok megléte esetén pedig a kiváltó ok lezajlása fogja lezárni az előző állapotot és vezet át a következőbe. A feliratok pontos formája a következő:

Átmenet ::= [Kiváltó ok][[Feltétel]]/[Tevékenység]

Kiváltó ok ::= Események[(Értékadások)]

Események ::= Esemény (,Esemény)*

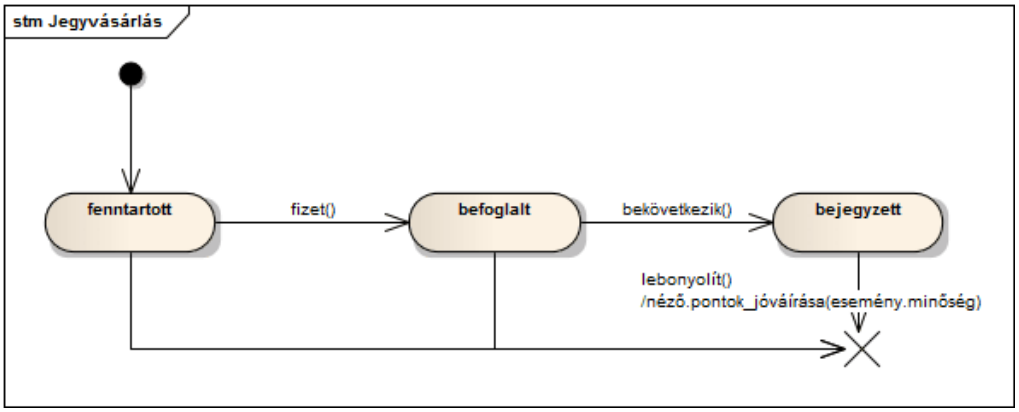
Értékadások ::= Értékadás (,Értékadás)*

Értékadás ::= Attribútum|Attribútum:Típus

Protokollátmenet ::= [[Előfeltétel]] Metódushívás/[[Utófeltétel]]

Átmenetszakasz ::= Feltétel

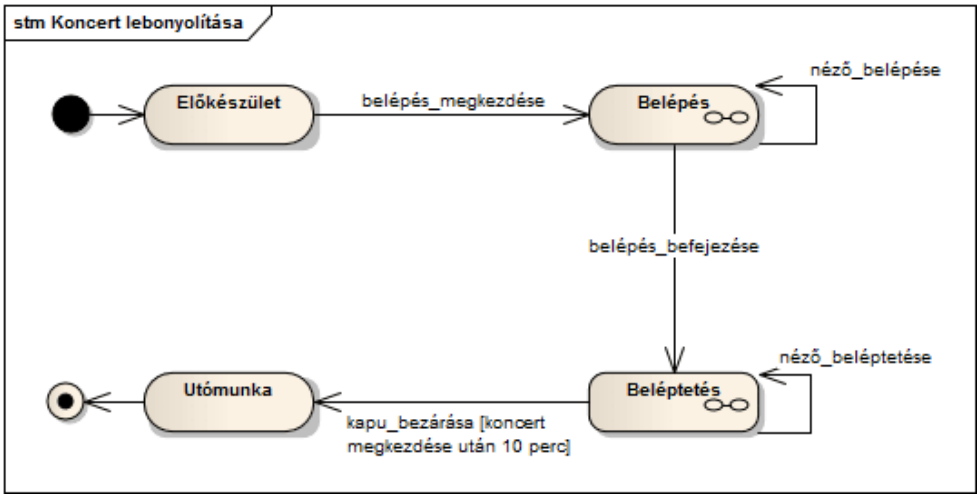
Az objektum életciklussal rendelkező osztályok esetén természetes lehet, hogy az objektum a működését befejezően megszűnik. Ezt a 8.3. ábrán látható módon a végállapot segítségével modellezhetjük.



8.3. ábra: Végállapot jelzése állapotdiagramon

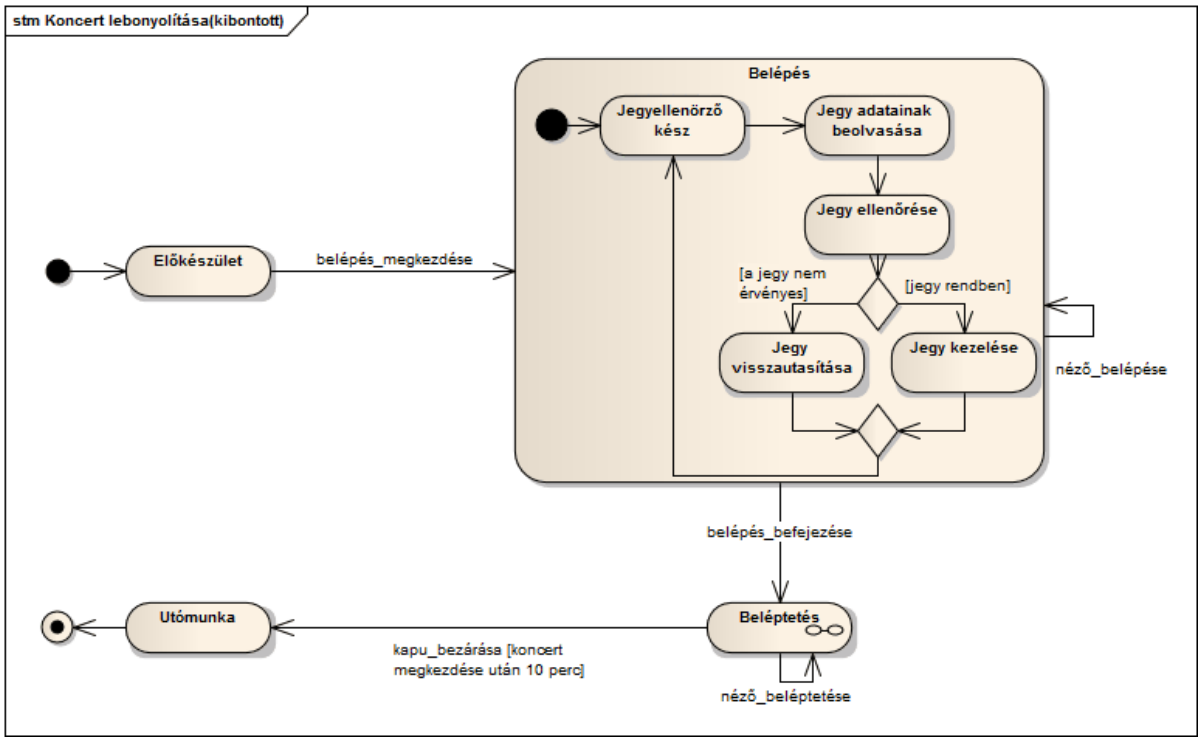
Az ábráról leolvasható, hogy a bónuszpontok csak a koncert lezajlása után kerülnek jóváírásra a rendszerben.

A 8.4-es ábrán a Koncert lebonyolításának objektum-életciklusa látható. A Néző ellenőrizteti a jegyét, belép a koncert helyszínére. A beléptetés a koncert kezdése után 10 percig folytatódik, aztán a kapukat bezárják.



8.4. ábra: A koncert lebonyolítása objektumainak életciklusa

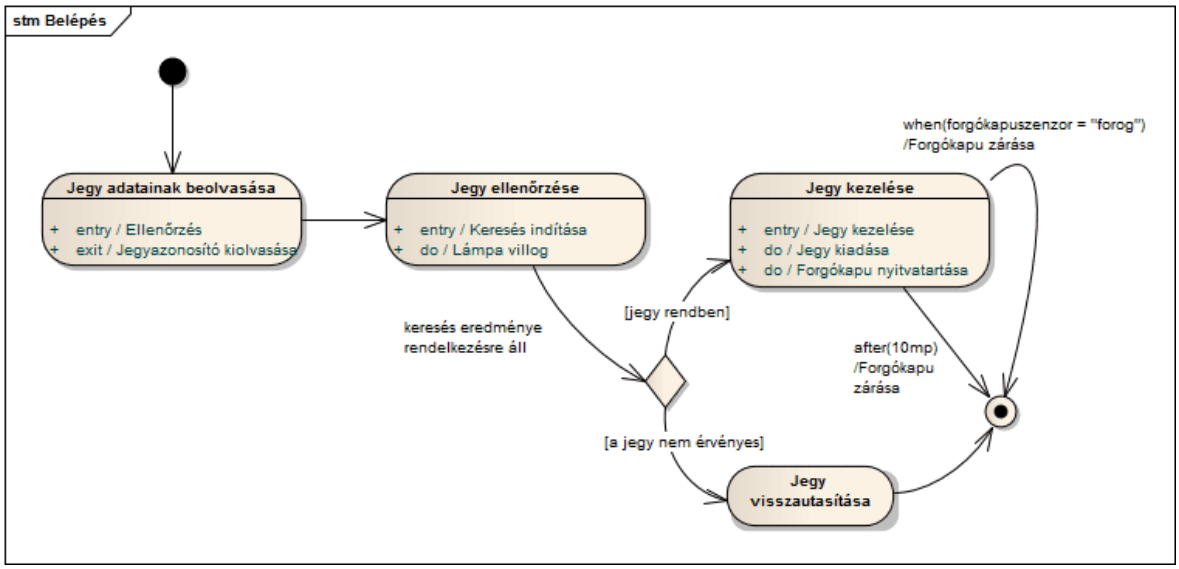
A feltételek a diagramon természetes nyelven jelennek meg, mivel itt még alapvetően az elemzési-tervezési szint szakterületi fogalmain van a hangsúly. A Belépés állapot sarkában látható jelölés arra utal, hogy egy összetett állapotról van szó, amely további régiókat és azon belül állapotokat és átmeneteket tartalmaz. A 8.5. ábra mutatja ennek kibontott formáját. A Belépés csak egy régiót tartalmaz. Ennek neve látható a keret tetején, a második részén pedig maga az állapotautomata. Az ilyen állapotot szekvenciális összetett állapotnak nevezzük. Ha a Belépés állapotba jut a folyamat, akkor a régió első állapotába jut az átmenet. Az utolsó állapot után a folyamat vagy a Belépés állapotába jut vissza vagy a következő állapotba jut a feltételeknek megfelelően.



8.5. ábra: A 8.4. ábra finomítása

A jegy ellenőrzése vezethet arra az eredményre, hogy a jegy nem érvényes a koncertre. Ezt a tevékenységdiagramoknál megismert esetválasztó csúccsal modellezhetjük. A feltételek szögletes zárójelben láthatóak az átmeneteken, amelyek az esetválasztó csúcsból indulnak ki. Ha az esetválasztó csúcsból kiinduló átmenetek feltételei kölcsönösen kizárják egymást, és lefednek minden esetet, akkor a feltételt az esetválasztó csúcsba is be lehet írni.

A 8.6-os ábra részletesebben mutatja meg a Belépés állapot Jegyellenőrzés részét. A tartalmazott állapotok két részre osztva jelennek meg: a felső rekeszben az állapot neve, az alsóban azok az események láthatók, amelyek az állapottal kapcsolatosak. Ezek esemény/tevékenység formában jelennek meg. Események lehetnek az objektum metódusai, illetve előredefiniált események. Ezek az entry, amely az állapotba történő belépést, a do, az állapotban való tartózkodást, és az exit, amely az állapotból való kilépést jelenti. Az ábrán jegy beolvasása állapotba történő belépéskor megtörténik az ellenőrzés, majd a kilépéskor a jegy azonosítószámának leolvasása. A jegy elfogadása állapotból két átmenet indul ki. Az egyiket az váltja ki amikor a forgókapu forog, amely egy változási esemény. Ennek argumentuma egy logikai kifejezés, és ha ez igazzá válik, akkor az esemény bekövetkezik, tehát ha a Néző a forgókaput forgatja a belépéssel. A másik átmenet egy időzített esemény. Ezek az események akkor indulnak el amikor az állapotátmenet abba az állapotba érkezik, és amint az idő lejár az állapotátmenet bekövetkezik. Ebben az esetben, ha jegy elfogadása állapotba érkezés után 10 másodpercig a forgókapu nem fordul el, akkor a forgókapu bezár.

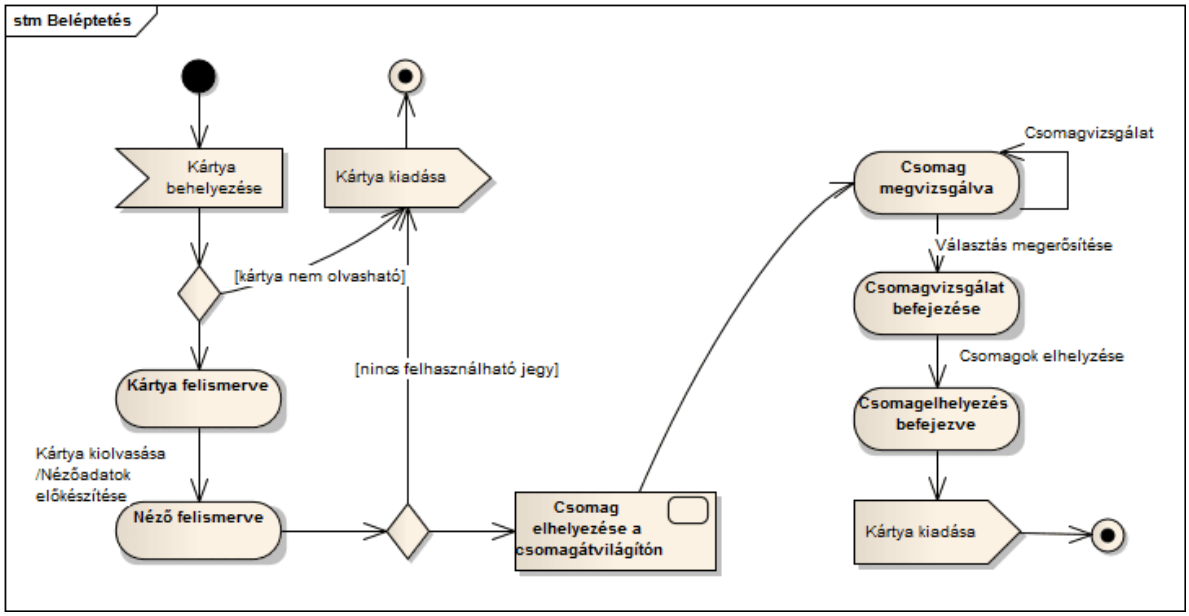


8.6. ábra: A Belépés állapotának kifejtése

Használati esetek életciklusa

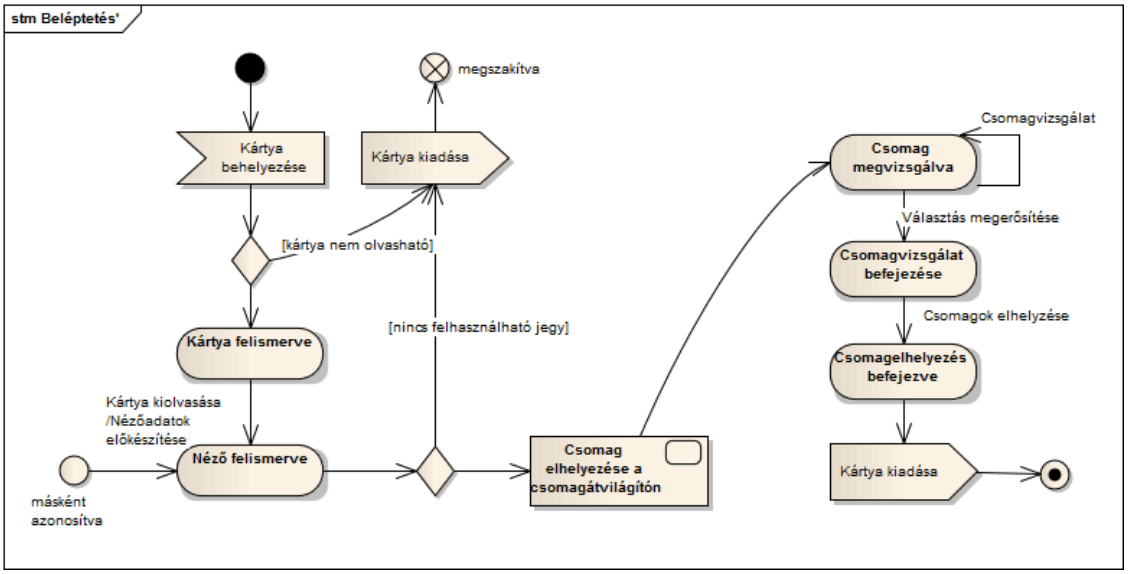
A használati esetek folyamat alapú leírásával találkozhattunk már a tevékenységdiagramok ismertetésénél is. A tevékenységdiagramok alkalmasabbak ezek modellezésére, mint az állapotautomaták, de szükség lehet a modellezés során ennek használatára, ha például előírásként jelenik meg a követelmények között. A Beléptetés során megjelenő állapotokat mutatja a 8.7-es ábra. Bejövő események jelzésére behajtott szélű négyszöget használ az UML.

Ezek az úgynevezett beérkezési események, amelyek az állapotátmenetet kiváltják. Az ábrán ilyen a kártya behelyezése esemény. Az állapotátmenetek által kiváltott eseményeket kétféleképpen jelölhetik. Az egyik a küldési tevékenység, amely nyílheggyel ellátott négyszögként jelenik meg, a másik a tevékenység, négyszöggel jelölve. Az ábrán a küldési tevékenységre a kártya kiadása, tevékenységre a csomagvizsgálat mutat példát.



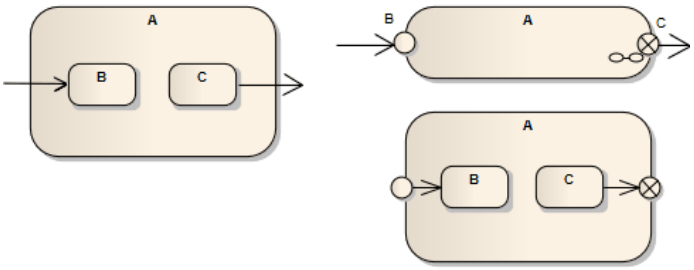
8.7. ábra: Beléptetés használati eset lefutása állapotautomataként

Előfordulhat, hogy az eszközök nem minden esetben működnek, így a bónuszkártya nem olvasható. Lehet olyan eset is hogy a Néző nem rendelkezik bónuszkártyával. Ilyenkor is biztosítani kell a beléptetést. Ilyenkor lehetőséget kell adni, hogy a Néző bónuszkártya nélkül is részt vehessen a csomagvizsgálaton. Erre a belépési és kilépési pontok alkalmasak. Ezek a pontok a komplex állapotra utaló be- és kilépési pontok. A kilépési pont bekarikázott x-ként jelenik meg, az ábrán a megszakítva állapot mutatja. A belépési pont egy kis kör, a komplex állapot határán.



8.8. ábra: Beléptetés állapotának továbbfejlesztése belépési és kilépési pontokkal

A 8.9-es ábrán a belépési és kilépési állapotok azonos értékű jelölése látható:



8.9. ábra: Belépési ás kilépési pontok jelölése komplex állapotok esetén

Protokollok

Az objektumdiagramnál megismertük az objektumok közötti együttműködéseket és az objektumok ezen belüli szerepköreit. Az egyik ilyen szerep, amivel gyakran találkozhatunk a protokollszerep. Láttunk, hogy az objektumok életciklusának leírására jól használhatók az állapotautomaták, ilyenkor azonban egy úgymond fehér doboz modell nézetből szemléljük az objektumokat, tehát ismerjük az objektum felépítését, belső állapotait is. Ha az a fontos, hogy a részleteket ne mutassuk meg a modellezés során, és csak arra koncentrálunk, hogy az objektum milyen kapcsolatokkal rendelkezik, hogyan érhető el, akkor a protokollszerepet használhatjuk. A protokollszerepek együttműködését protokolloknak hívjuk.

A protokollszerep használatával tulajdonképpen erősödik az objektumorientáltság, az objektumok határai meghatározásra kerülnek. Egységként kezeli az objektumokat, és a viselkedését kívülről figyelhetjük meg. Ebben a nézetben az összekötők és a csatlakozók játszák a főszerepet.

8.10. ábra: A protokollszerep

A protokollszerepek leírására a protokollautomata használatos. Ez tulajdonképpen az állapotautomaták egy specifikus változata. A protokollautomatáknál az állapotátmenetek helyett protokollátmenetekről beszélünk. A protokollátmeneteknek nincsenek hatásai, az állapotátmeneteket elő- és utófeltételekkel írjuk le. Nyelvtana következő:

Átmenet ::= [Kiváltó ok][[Feltétel]]/[Tevékenység]

Kiváltó ok ::= Események[(Értékadások)]

Események ::= Esemény (,Esemény)*

Értékadások ::= Értékadás (,Értékadás)*

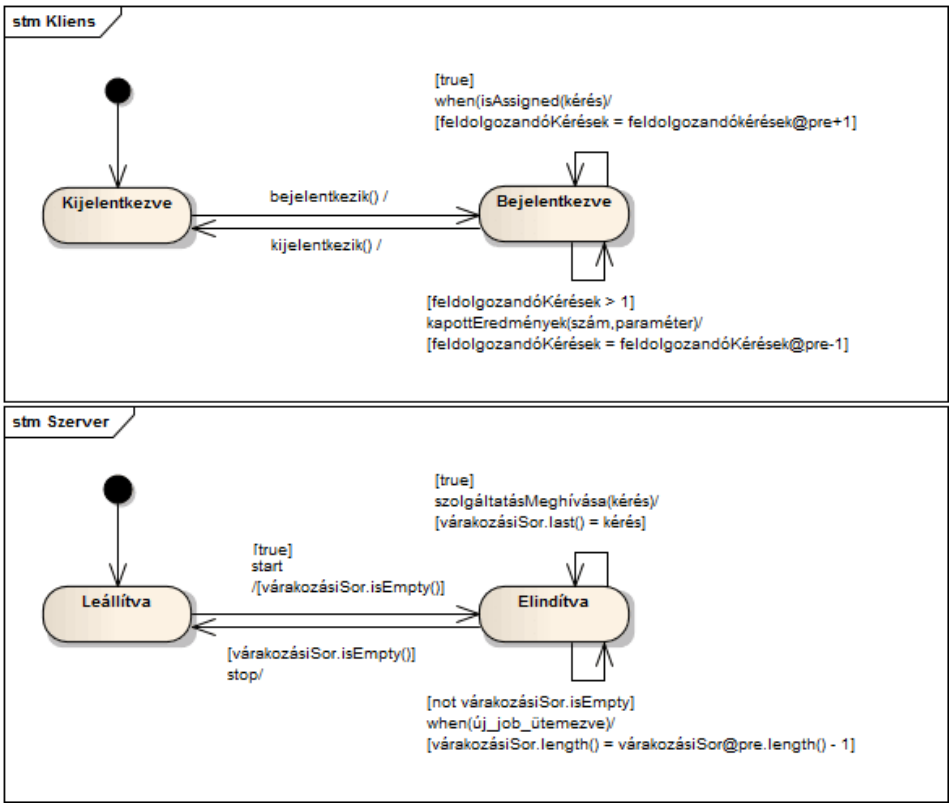
Értékadás ::= Attribútum|Attribútum:Típus

Protokollátmenet ::= [[Előfeltétel]] Metódushívás/[[Utófeltétel]]

Átmenetszakasz ::= Feltétel

Az átmenetet kiváltó esemény az osztály egy publikus metódusa lesz, amelynek elő- és utófeltételei az átmenet elő- és utófeltételei lesznek. A kiváltó eseményt a protokollszerep határozza meg. A protokollautomaták annyiban különböznek még a hagyományos állapotautomatáktól, hogy a tulajdonképpeni állapoton kívül az automata rendelkezhet még egyéb értékekkel is, ami kívülről közvetlenül nem látható és nem módosítható. Ilyen a kliens/szerver protokoll esetében az, hogy a szervernek van egy várakozási sora, amelyben a fel nem dolgozott kérések jelennek meg. Ez természetesen hatással van a feldolgozásra és az állapot része, de nem lehet rá hatással lenni.

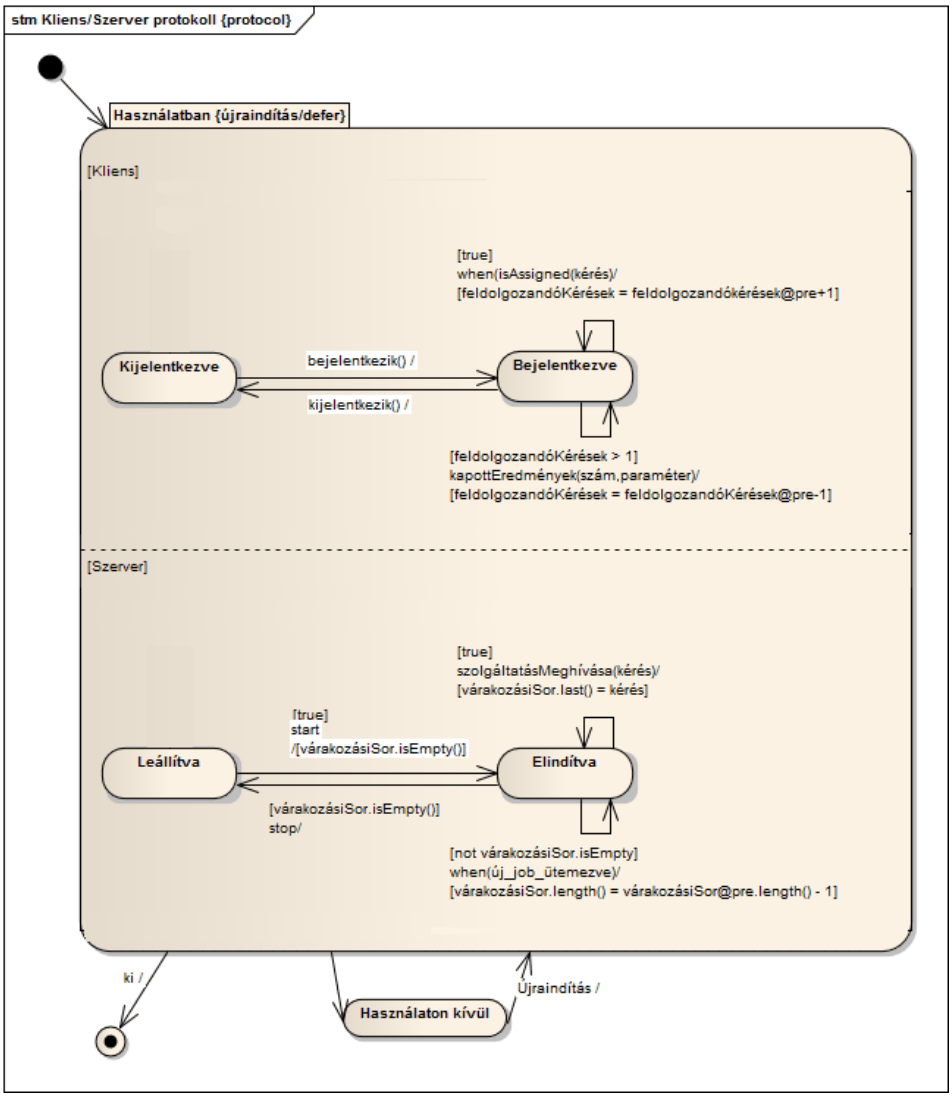
Példaként nézzük a kliens/szerver protokollt, amelynek működése protokollautomata segítségével részletesen bemutatható. Az architektúrában a kliens meghívja egy szerver szolgáltatásMeghívása műveletét. Paraméterként a saját azonosítója és egy paraméter adódik át. A szerver fogadja a kérelmet, puffereli azt, és visszatérési értékként a kérelemnek adott megbízási sorszámot adja vissza. A második lépésben a megbízást kiveszi a sorból, feldolgozza és a kapottEredmények műveleten keresztül átadja az eredményt a megbízónak. Ehhez csatolja a megbízási számot. A protokollautomatán kívülről ebből természetesen a belső műveletek nem láthatóak, a kliens és a szerver egymással való kapcsolata csak a műveleteken keresztül jeleníthető meg. A 8.11. ábrán a szerver kiszolgálja a klienskérést.



8.11. ábra: Kliens/szerver protokollautomata (RequireService és ProvideService) [6]

[6] Harald Störrle: UML 2

A két protokollszerep természetesen párhuzamosan fut egymással. Ez párhuzamos állapoton keresztül ábrázolható, amely több egymástól független régióra tagolódik (8.12. ábra). Minden szerephez tartozik egy régió. A párhuzamos régiók egymástól függetlenül, egyidejűleg futnak le, ortogonálisak. A párhuzamos állapothoz érkezéskor minden régióban egyszerre indul el a feldolgozás és csak akkor lép a következő állapotba, ha minden régió végzett. Ha egy ilyen összetett vagy egy egyszerű állapot befejeződik akkor végesemény váltódik ki. A végesemény csak akkor váltódik ki, ha semmilyen más esemény nem váltódik ki az állapotban.



8.12. ábra: Kliens/ szerver protokoll ortogonális régiói

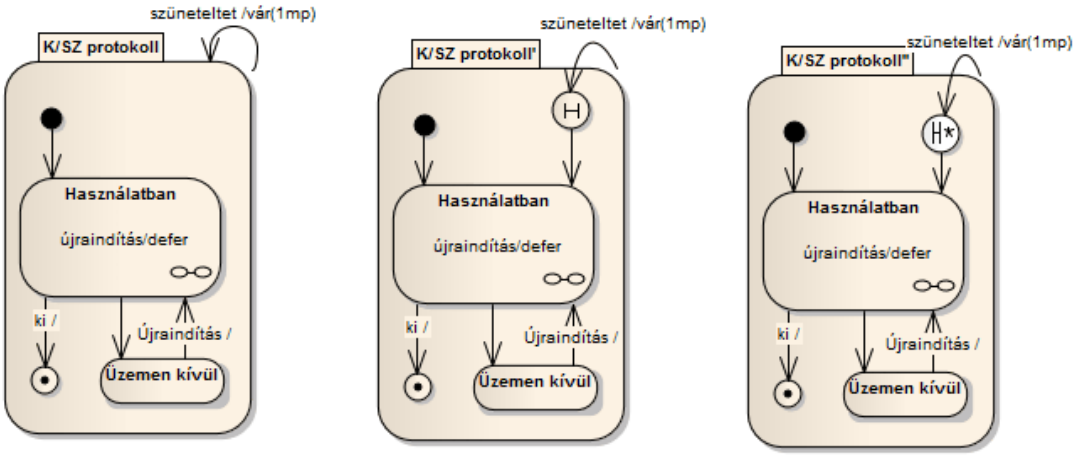
8.13. ábra: B összetett állapot különböző lezárása esetén elérhető állapotok

A 8.13-as ábrán látható, hogy ha a B állapot normál módon fejeződik be, akkor a végesemény kiváltódik és az A állapotba kerül a vezérlés, viszont ha a X esemény bekövetkezése miatt fejeződik be a B állapot, akkor a C-be jut a vezérlés.

A kliens/ szerver protokoll esetében ez úgy néz ki, hogy ha mindkét régió esemény nélkül lefut, akkor a használaton kívüli állapotba fut be a vezérlés, ha viszont a „ki” esemény kiváltódik, akkor a végállapotba jutunk. A használaton kívüli állapotból akkor jutunk vissza a használatban állapotba, ha az újraindítás esemény bekövetkezik.

Történeti állapotok

Lehetnek olyan esetek, amikor egy állapotot nem megszakítani szeretnénk, hanem csak szüneteltetni. Ha egy ki esemény váltódik ki és mondjuk pár másodperc után visszatérünk az állapotba, akkor az állapot végrehajtási állapota törlődik, és visszalépéskor a protokoll újraindul. Ennek a problémának a megoldására került bevezetésre a történeti állapot. A történeti állapotokból kétfajtát ismerünk: egyszerű történeti állapotot és a mély történeti állapotot. A történeti állapot jelölésére egy bekarikázott H betűt helyeznek el az összetett állapot jobb felső sarkában (8.14. ábra).



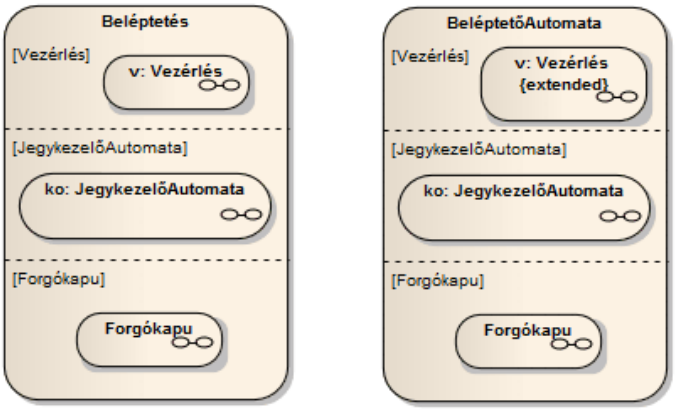
8.14. ábra: Történeti állapotok

Az egyszerű történeti állapot az állapotkonfiguráció felső szintjét őrzi meg. Ez azt jelenti, hogy az állapotba történő visszalépéskor a szüneteltet esemény lejártával az üzemben illetve az üzemen kívüli állapotok maradnak meg, az üzemben alállapotai azonban nem. Ha erre van szükségünk, akkor a mély történeti állapot használata javasolt. Jelölése bekarikázott csillaggal ellátott H betűvel történik. A mély történeti állapot alkalmazásával az állapotban minden állapotkonfiguráció megőrződik.

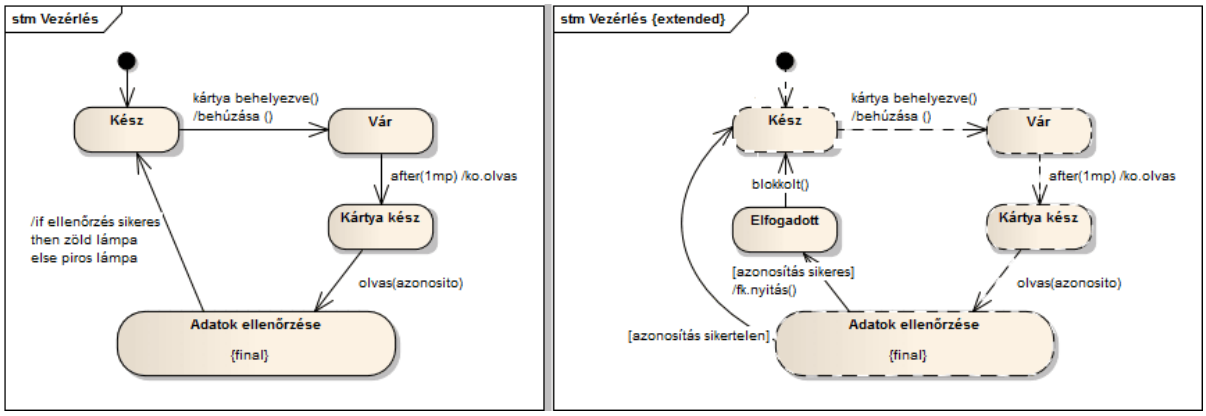
Állapotautomaták specializációja

A rendszerek futása során vannak olyan esetek, amelyek közös pontokat, eseményeket, állapotokat tartalmaznak. Egy ilyen átfedő eset lehet, ha a beléptetést a jegykezelő automata hibája okán a személyzet egy tagja végzi el. A jegykezelő automata normális működése és a személyzet segítségével történő jegykezelés esetén is lesz egy jegyolvasó automatánk, amely beolvassa a jegy sorszámát. A személyzet segítségével történő jegykezelésnél lehetséges, hogy nincs szükség forgókapu használatára. Látható, hogy vannak átfedések a két folyamat között. Ilyen

esetekben a redundanciák kiküszöbölésére érdemes az átfedéseket egyszer modellezni, és ezt a modellt felhasználni mindkét változatnál. A 8.15-ös ábrán láthatóak a Személyzeti pult és a Jegykezelő automata objektumai. A Jegyolvasó automata itt kétszer jelenik meg. A jelölés arra utal, hogy mindkettő ugyannak a jegyolvasó típusnak a példánya. A JegykezelőAutomata::v állapot esetében egy {extended} jelölést látunk az ábrán. Ez azt jelenti, hogy az itt megjelenő objektum egy kiterjesztése a Vezérlés állapotautomatának. A kiterjesztés az eredeti állapotautomata egy specializációját jelenti.



8.15. ábra: Automata és személyzet által végzett jegykezelés viselkedését leíró állapotautomaták



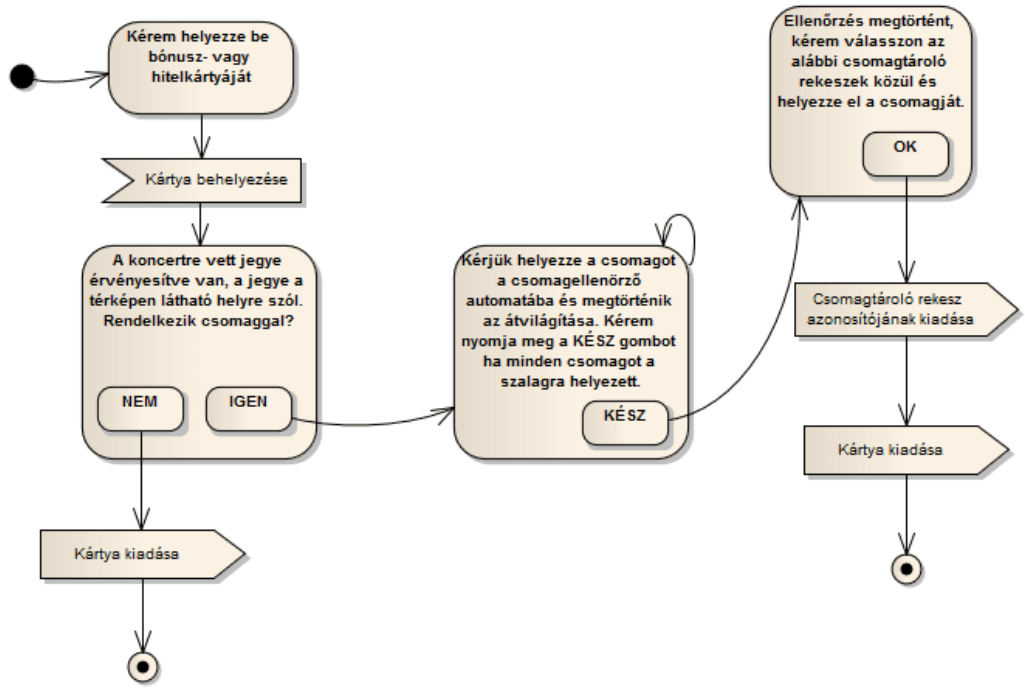
8.16. ábra: Jegykezelést és automata jegykezelést leíró állapotautomaták rendszerképe

A Vezérlés állapot részletes megjelenítésében (8.16. ábra) látható, hogy a kiterjesztett állapot az általános állapottal szemben több állapotot tartalmaz. A kiterjesztés során nem változtatott állapotokat és állapotátmeneteket szaggatott vonallal jelölik. Ha olyan állapot, régió, vagy állapotautomata van a rendszerben, amely nem kiterjeszthető, azt a final tulajdonságértékkel kell ellátni.

Dialógusalapú párbeszéd modellezése állapotautomatával

A felhasználókkal történő párbeszéd és a felhasználói felület dialógusainak leírása is lehetséges állapotautomatákkal. A dialógusablakok közötti átlépések lesznek ilyen esetben az állapot-

átmenetek, a menük pedig maguk az állapotok. Ez a modell segítheti a végfelhasználóval történő egyeztetést, valamint az interjúztatásban is alternatíva lehet. A 8.17-es ábra a Csomagkezelő-automata menüinek használatára mutat példát.



8.17. ábra: A csomagkezelés dialógusának vázlata

Az állapotautomatákat táblázatos formában is le lehet írni (8.1. táblázat). Ebben az esetben az állapotátmenetek, az állapotok illetve az átmenetek számára készül külön táblázat. Erre akkor lehet szükség, ha valamilyen okból a modell felhasználóinak gondja van a grafikus megjelenítéssel, vagy a követelményekben ez a forma szerepel.

8.1. táblázat Egy állapotautomata táblázatos megjelenítésének formátuma

Átmenetek					
Név	**Kiváltó ok**	**Előfeltétel**	**Tevékenység**	**Utófeltétel**	**Megjegyzés**
1	lebonyolít()	-	-	-	-
2	fizet()	-	-	-	-
. . .					

Állapotok					
Név	**Típus**	**Alállapotok**	**Invariánsok**	**Tevékenység**	**Megjegyzés**
Indulóállapot	belépési pont				
Fenntartott	egyszerű állapot				
. . .					

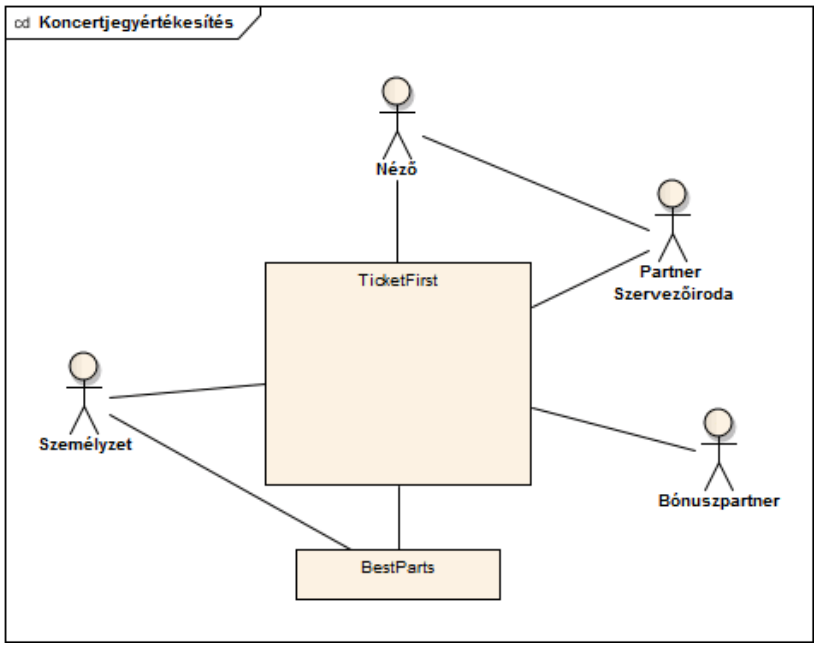
Állapotátmenetek				
Induló\Cél	**Indulóállapot**	**Fenntartott**	**Befoglalt**	**. . .**
Indulóállapot	-	1	-	
Fenntartott	-	-	2	
Befoglalt	-	-	3	
. . .				

9. A rendszer architektúrája és komponensei: kontextus-, szakarchitektúra- és montázsdiagram

A használati esetek és az üzleti folyamatok azonosításának egyik célja volt a rendszerben szereplő és a rendszerrel kapcsolatban álló aktorok azonosítása. A rendszer architektúrájának és felépítésének megismerésével a cél szintén a rendszer környezetének megismerése, de most strukturális nézőpontból. Erre az UML-ben több diagram is rendelkezésre áll. Ezek a már megszokott módon más szemszögből nézik a modellezendő architektúrát. Így találkozhatunk kontextus-, szakarchitektúra-, montázs-, csomag-, komponens-, rendszerstruktúra- és kihelyezési-diagramokkal. Ahogy látjuk a modellezési eszköztár széles, már csak meg kell találnunk, hogy a projekt részvevőinek igényét melyik vagy melyek elégítik ki.

Kontextusdiagram

A TicketFirst rendszer környezetének felmérésére egyik alkalmazható diagramtípus a kontextusdiagram (9.1. ábra). Az eddig megismert résztvevők alapján biztosan lesznek olyan alrendszerek vagy külső rendszerek, amelyekkel a TicketFirst kapcsolatban áll. Ezek a rendszerrel együtt alkotják az Automata koncertjegyértékesítés összrendszert. Az elemei a TicketFirst, a szomszédos rendszerek, a felhasználók és egyéb résztvevők. A rendszereket négyszögek jelölik, a személyeket és csoportokat pedig pálcikafigurák mint az üzleti folyamatok és használati esetek aktorait. A rendszerek jelen esetben úgy viselkednek, mint az osztályok, az UML-ben a Classifier speciális esetei lesznek. Míg a rendszereknek ismerhetjük a viselkedését és a tulajdonságait a modellben, addig az aktorok jellemzői ilyen szempontból nem kerülnek kifejtésre.



9.1. ábra: A TicketFirst rendszer kontextusdiagramja

A 9.1-es ábráról a következő olvasható le:

- Van egy BestParts nevű rendszer, amelyben a partnerek adatai vannak tárolva. Ez kapcsolódik a TF rendszerhez
- A rendszerrel szoros kapcsolatban álló aktorok: Személyzet, Partner Szervezőiroda/ /Jegyiroda, Néző/Vásárló, Bónuszpartner. A projekt csak ezekre koncentrál, tágabb kör nem feladata
- A vásárlóknak közvetlen kapcsolata kell legyen a TicketFirst egyes alrendszereivel, például a jegyvásárlás alrendszerrel
- Nincs betervezve közvetlen együttműködés az érdekeltek között. A bónuszpartnerekkel és a partner szervezőirodákkal a nézők is felvehetik a kapcsolatot.

Az aktorokat és feladataikat, kapcsolatukat a rendszerrel táblázatos formában is le lehet írni. Erre példa az alábbi táblázat.

9.1. táblázat: A rendszerrel kapcsolatban álló aktorok jellemzői

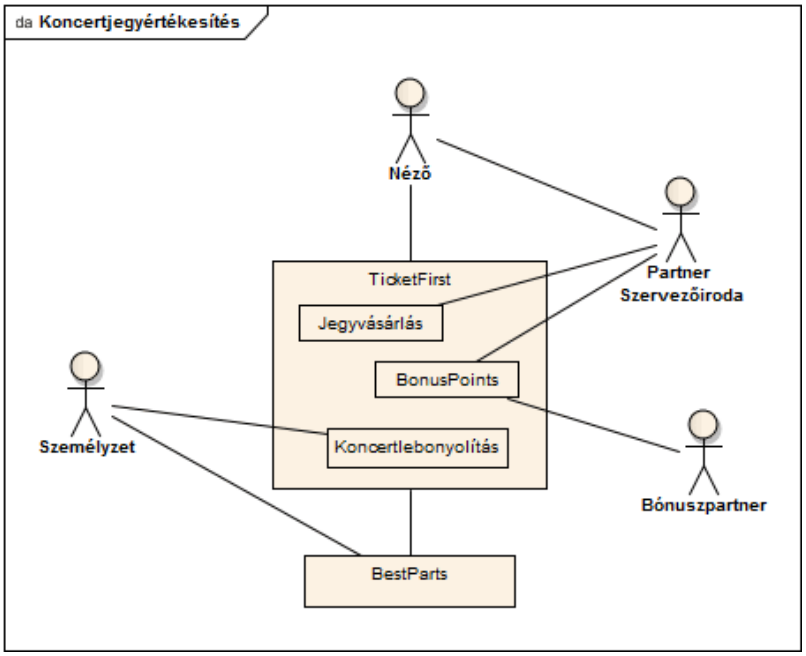
Személyzet	
Feladat	nézők segítése a belépésnél és jegykezelésnél
Mennyiség	1:2000-hez a személyzetnek a nézőkhöz viszonyított aránya
Fajta	a TicketActual alkalmazottai
Betanítási idő	fél nap

Partner szervezőiroda/jegyiroda	
Feladat	jegyek értékesítése, programok szervezése
Mennyiség	jelenleg 20
Fajta	Önálló vállalkozások
Betanítási idő	-

Bónuszpartner	
Feladat	Bónuszpontok felírása, felhasználása
Mennyiség	jelenleg 40, későbbi tervek 70
Fajta	Önálló vállalkozások
Betanítási idő	-

Szakarchitektúra-diagram

A nagyobb rendszereknél lehet szükséges a szakarchitektúra használatára, ha további részekre szeretnénk tagolni a rendszert. Ilyen alrendszer a Jegyvásárlás, a Koncertlebonyolítás és a BonusPoints alrendszerek. Az előző kontextusdiagram tehát az alábbi szakarchitektúra-diagrammal finomítható tovább (9.2. ábra):



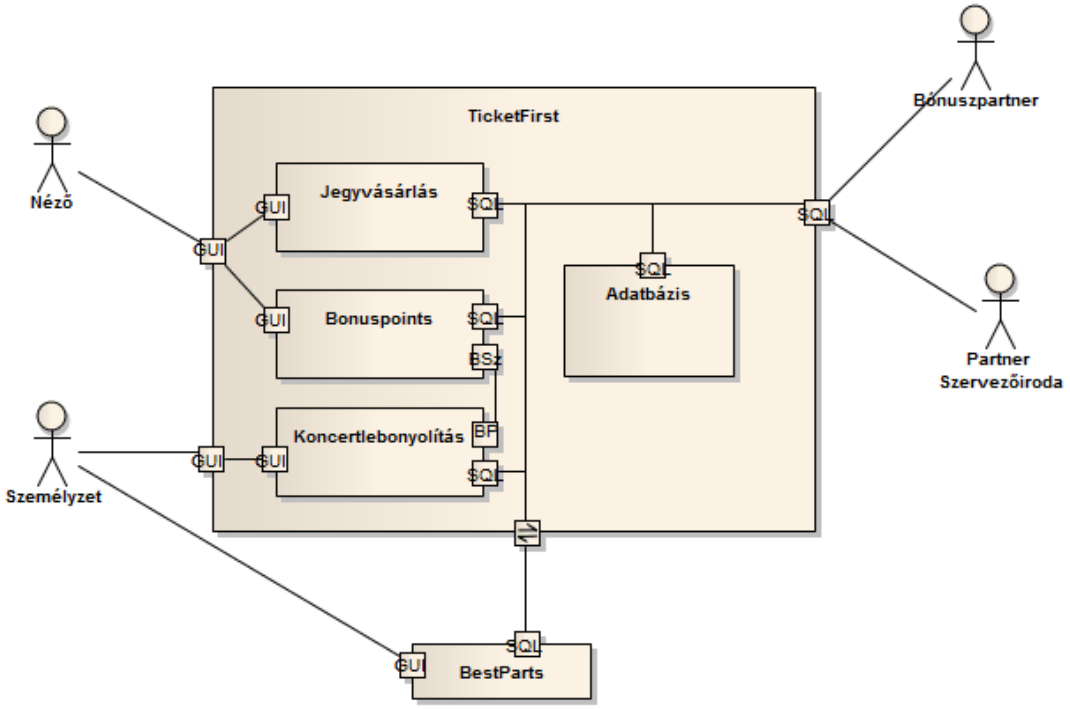
9.2. ábra: A TicketFirst szakarchitektúrája

A Partner szervezőiroda a Jegyvásárlással és a Koncertlebonyolítás alrendszerekkel lesz kapcsolatban, a Bónuszpartner pedig a BonusPoints alrendszerrel. Tehát ezen a diagramon már azonosítani lehet, hogy az aktorok mely alrendszerre lesznek hatással, melyikkel lesznek közvetlen interakcióban. A Néző az egész TicketFirst rendszerrel kapcsolatban van, nincs hozzá azonosítva külön alrendszer. Ez azt jelenti, hogy ő a rendszer minden alrendszerével együttműködik, a diagram átláthatóságának megőrzése érdekében nem szükséges a közvetlen összekötők megrajzolása.

Montázsdiagramok

A montázsdiagramok nevének szó szerinti fordítása angol nyelvről az összetett struktúra diagram. A struktúra nem más, mint egymáshoz kapcsolódó futási idejű elemek egy konfigurációja. Ez jobban kifejezi azt, hogy mi is az amit ezzel a diagramtípussal le lehet írni: egy rendszer, illetve összetett osztályok felépítését írják le. Architektúradiagramnak is nevezik őket.

A kontextusdiagramból kiindulva megalkothatjuk a rendszer montázsdiagramját. Ebben a nézetben már csatlakozókkal és összekötőkkel találkozhatunk az egyes részek határain. A csatlakozók kis fehér négyzetek, az összekötők pedig a köztük lévő vonalak. A csatlakozók egyik típusa a relécsatlakozó, amely az áthaladó jeleket csak továbbítja. Az előző diagramból kiindulva azonosíthatóak a rendszer alrendszerei. Ezek maguk is összetett struktúráknak tekinthetők. Az alrendszereken kívül találhatunk még egy adatbázist a rendszerben, amelyet az alrendszerek közösen használnak. Az összetett struktúrák tovább bonthatóak, és a részeik maguk is struktúrák lehetnek. A Koncertlebonyolítás alrendszer a SzemélyzetiPult, a BeléptetőAutomata, a CsomagellenőrzőAutomata és az Infrastruktúra modulokból áll. Ezek legtöbbje szintén tovább bontható.



9.3. ábra: A TF rendszer montázsdiagramja

A 9.3. ábran látható csatlakozók az osztályok interakciós portjai. Feladata, hogy az osztályt egységbe zárja, és biztosítsa a környezettel történő interakciót. Az osztály belsejének mindeközben rejtve kell maradnia, tehát a csatlakozónak minden szerepkört el kell látnia, amit az osztály lefed. A csatlakozó tehát a kínált mellett a használt interfészt is magában foglalja, valamint a használati protokollt, a funkcionalitást. Tulajdonképpen a csatlakozó is egy osztálynak tekinthető az UML-ben de struktúra nélküli osztály. Az interfésznél jóval több funkciót lát el. A csatlakozót az osztály keretén elhelyezett kis négyzet jelöli. Az alábbi csatlakozófajtákkal találkozhatunk:

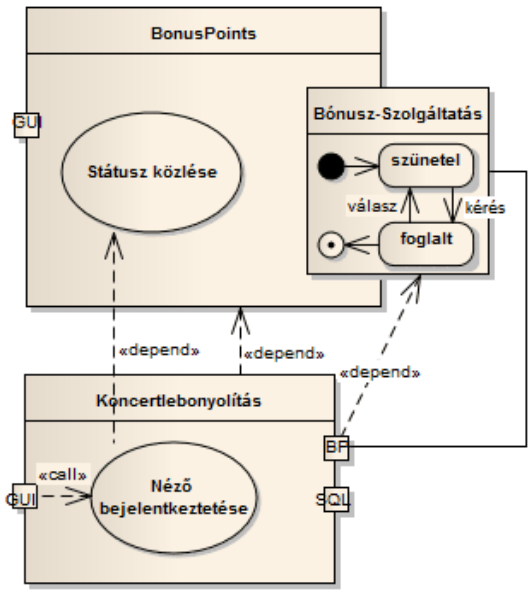
- Belső csatlakozó: a Koncertlebonyolítás GUI csatlakozója a TF egy kontextusával van kapcsolatban, a személyzet aktorral. A belső csatlakozó a TF.Koncertlebonyolítás.GUI.
- Relécsatlakozó: megakadályozza, hogy a belső csatlakozókat kívülről látni lehessen. Delegáló összekötőkkel együtt használják. Az ábrán relécsatlakozó a TF.SQL
- Transzpondercsatlakozó: a kontextus és a rész csatlakozói között áll és önálló funkciót lát el. Ilyen pl. az TF.GUI, amely a TF.Jegyvásárlás.GUI és a TF.BonusPoints.GUI elérést fogja össze

A csatlakozók összekötőkön keresztül kapcsolónak egymáshoz. Az összekötők vonalakat jelentnek a csatlakozók között. Nincs rajtuk felirat. Hogy két rész között legyen e összekötő, azt a tervező feladata eldönteni. Vannak összetett összekötők is, amelyek saját viselkedéssel bírnak, például pufferelés, késleltetés hiba stb. Ezek a részletek állapotautomatákkal modellezhetők.

Eddig az UML-ben kétfajta kapcsolattípust ismertünk meg, az irányított és az irányítatlan asszociációt. Az irányított asszociáción belül ismerjük a magábanfoglalást, a kiterjesztést és az általánosítást. Az irányított asszociációk egy újabb formája a függőség, amely a szolgáltatást

használó és a szolgáltatást nyújtó közötti nem specifikált kapcsolatot jelöl. A szolgáltatást használótól a szolgáltatást nyújtó felé mutató szaggatott vonalú nyíl jelzi. A nyílon a depend szó jelzi a függőséget.

A 9.4. ábra látható a Koncertlebonyolítás és a BonusPoints közötti kapcsolat. A köztük lévő függőséget az S-Kliens (státusz-kliens) és a S-Szolgáltatás (státusz-szolgáltatás) közötti függőség határozza meg, hiszen a szerepköröket ők valósítják meg. Mindkettő a Jegyellenőrzés és a Státusz közlése használati esetek közötti függőségből származik.



9.4. ábra: Csatlakozók viselkedése és függőségi kapcsolat modellezése

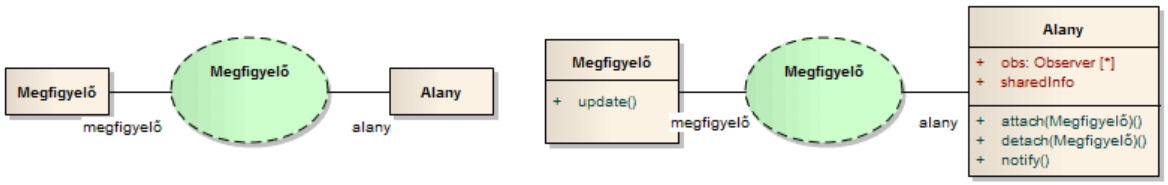
Együttműködések

Az együttműködések a modellezés során egy olyan rendszert írnak le, amelyben a résztvevők egy közös cél érdekében dolgoznak együtt. Ez a közös cél egy funkcionalitás megvalósítása. Ebben az együttes munkában a résztvevők bizonyos szerepeket vállalnak magukra, úgynevezett szerepköröket vesznek fel. Ezekkel a szerepkörökkel találkozhattunk az objektumdiagramok tárgyalásánál. Ilyenkor a modell leszűkül az éppen aktuális funkció elérése érdekében a fontosabb jellemzőkre. Az együttműködéseket a tervezési mintákhoz, az architekturális stílusokhoz és a kontextus-együttműködésekhez hozták létre. A megfigyelő minta egyértelműen leírja az résztvevők egy célért történő együttműködésének lényegét.

Tervezési minták

A megfigyelő minta egy olyan struktúrát hoz létre, amelyben egy résztvevőhöz csatlakozva az összeshez elér a megosztott információ anélkül, hogy páronként kellene kapcsolatokat kialakítani. Ezzel csökkenteni lehet a kommunikációs vonalak számát és a terhelés is kisebb az egyes résztvevőknél. Ez a megfigyelő minta egy irodalmi példának tekinthető.

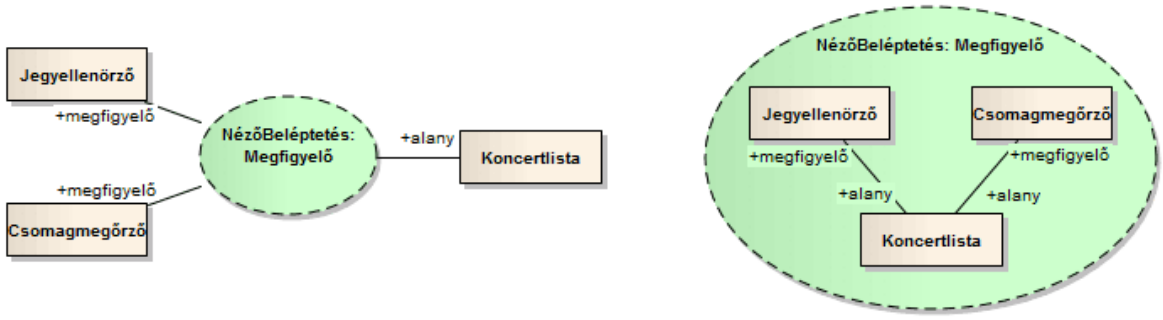
A megfigyelő minta általános sémája látható az alábbi ábrán, baloldalon. Ebben csak a résztvevők, és a mintában betöltött szerepköreik kerülnek feltüntetésre. A jobb oldali képen a szerepköröknél már attribútumok és metódusok is szerepelnek. Természetesen ezek a tulajdonságok szoros kapcsolatban állnak a jelen együttműködési struktúrával, valamilyen szerepet játszanak benne.



9.5. ábra: A megfigyelő minta struktúrája

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. Erre példa egy táblázatkezelő, amelynek az adatait lehet táblázatos formában munkalapon is szemlélni, vagy kör, esetleg oszlopdiagramon megjeleníteni. Ha bármelyik változik, az összes többinek automatikusan változnia, frissülnie kell. Ezt az adatfrissítést – amely nem más, mint a sharedinfo attribútum – látja el az Alany objektumunk – a notify művelettel – a megfigyelőmintában és a hozzá kapcsolódó megfigyelők tekinthetőek a különböző diagram vagy táblázattípusoknak.

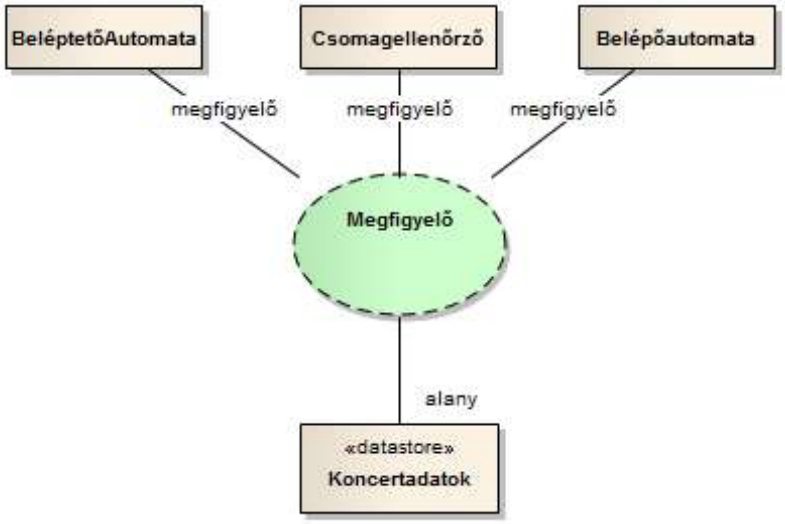
Egy együttműködés konkrét előfordulását együttműködés-előfordulásnak nevezzük. Ilyenkor az együttműködés minden szerepköréhez konkrét résztvevőket határoznak meg. A Megfigyelő szerepköröket a „Jegyellenőrző” és a „Csomagmegőrző” töltik be, az Alany pedig a „Koncertlista”. Az együttműködés-előfordulás ábrázolása látható a következő diagramon. Itt fel kell tüntetni a konkrét előfordulás nevét is. A részvevők közötti kapcsolatok szaggatott vonallal vannak jelölve. Egy tervezési minta leírása tovább részletezhető például interakciós diagramokkal vagy tervezési osztálydiagramokkal.



9.6. ábra: Példa konkrét együttműködés-előfordulásra

Architekturális stílus

Az architekturális stílus vagy más néven architekturaminta a tervezési mintához hasonlóan épül fel, de az együttműködésben nem osztályok és objektumok, interfészek és kapcsolatok vesznek részt – az architektúrából következően – hanem részek, csatlakozók és összekötők. A TicketFirst rendszer kontextusára épülő mintát láthatunk a 9.7. ábra.



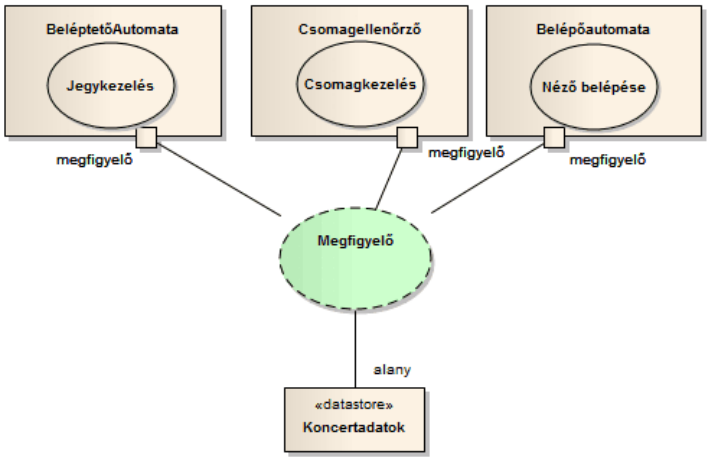
9.7. ábra: Architekturális minta

Koncertadatok: tartalmazza a koncertre jegyet vásárolt nézők adatait.

Beléptetőautomata: azonosítja és ellenőrzi a jegyet, hogy érvényes-e a koncertre, illetve ha van a Nézőnek bónuszkártyája, a nézőt azonosítja és jegyzi a belépést

Belépőautomata: a Néző csomagját ellenőrzi csomagátvilágítással, jegyzi, hogy ha tartozik hozzá csomag, illetve hogy szükség van e csomagmegőrzőben történő elhelyezésre

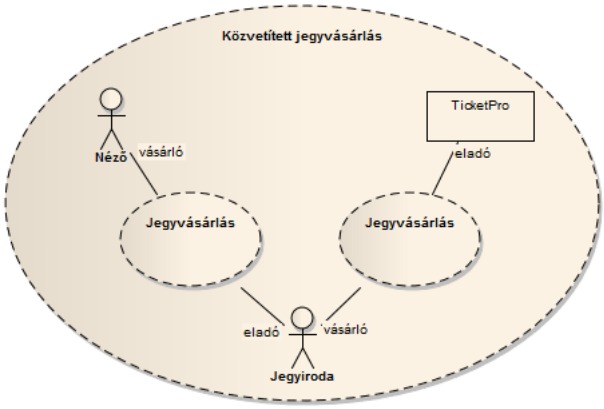
Csomagmegőrző: csomagok elhelyezése a megőrzőben, a jegyhez vagy a kártyához hozzárendelik a csomagmegőrző sorszámát.



9.8. ábra Az együttműködésben résztvevő szerepek azonosítása csatlakozókkal és használati esetekkel

Kontextus együttműködés

Még egy szinttel feljebb lépve a rendszer szintjén megvizsgálhatjuk az aktorok és a rendszerek közötti együttműködéseket. Erre való a kontextus-együttműködés. A rendszerszintre is jellemző, hogy egy résztvevő több szerepkört is betölthet, így például egy partner Jegyiroda lehet szolgáltató és ügyfél is egyidejűleg, attól függően, hogy melyik kontextusban tekintjük.



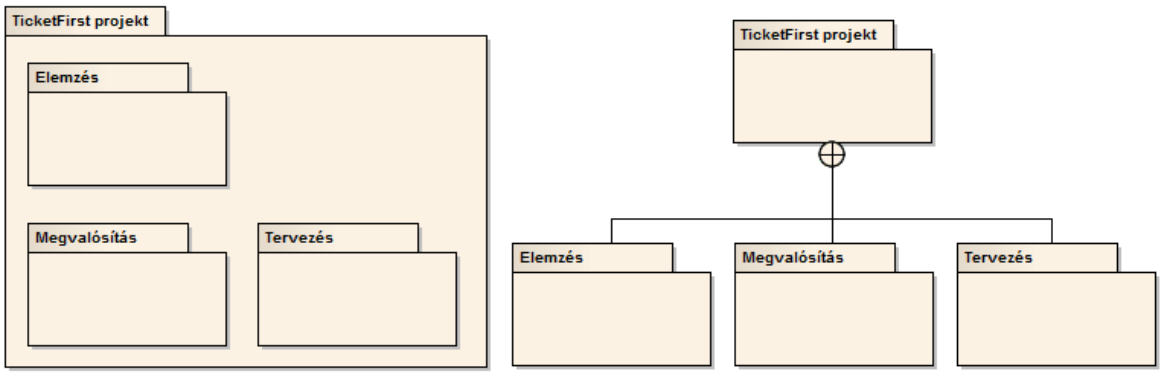
9.9. ábra: Jegyvásárlás együttműködési diagram rendszerszinten

10. A rendszer architektúrája és komponensei: csomagdiagram, komponensdiagram, kihelyezési diagram

Folytatva a rendszer strukturális modellezésnek megismerését a fejezetben a csomagdiagramról, a komponensdiagramról és a kihelyezési diagramról lesz szó. Ahogy megszoktuk, itt is mindegyik egy más-más eszközrendszert, nézőpontot képvisel. A csomagdiagramok az elemek csoportosításán alapulnak, a komponensdiagram a komponensorientált szoftverfejlesztésből átvett és átdolgozott diagramtípus, a kihelyezési diagram pedig ahogyan a neve is utal rá a rendszer telepítéséhez kapcsolódik szorosan.

Csomagdiagramok

Az elemek csoportokban való kezelésének, egy névtérben való egyesítésének illetve fastruktúrába való rendezésének a megoldására alkalmasak a csomagdiagramok. A csomagok a vizuális ábrázolásban mappaként jelennek meg. A csomag neve a mappán van feltüntetve. A csomag alkotóelemeit magában a csomagban vagy egy birtoklási kapcsolattal jelöljük, hasonlóan a kompozíciós kapcsolathoz.

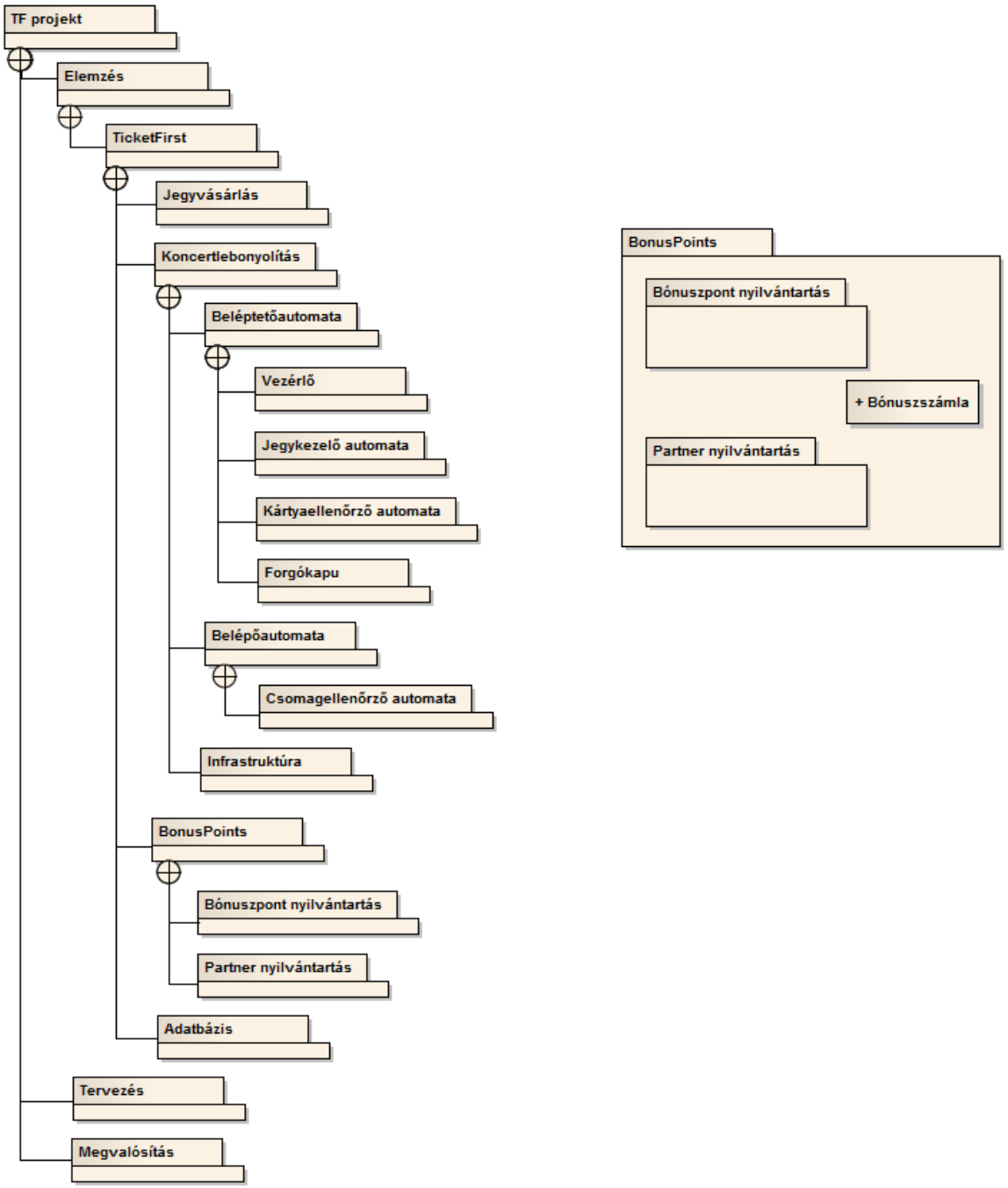


10.1. ábra: Csomagok vizuális ábrázolásának lehetőségei

Nagyobb rendszerek esetében a csomaghierarchiák használata is jellemző. A fastruktúrán belül minden elemnek minősített neve van. Ez a minősített név az elemet egyértelműen azonosítja. A név alakja a következő:

gyökércsomag ::(csomagnév::)*elemnév

Itt a gyökércsomag a hierarchia legmagasabb szintjén álló csomagot jelöli.



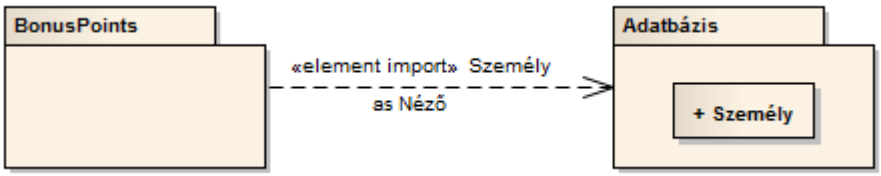
10.2. ábra: A csomagok aggregációhierarchiája

A hierarchia alapján a Bónuszszámla osztály minősített neve a következő:

TF projekt::Elemezés::TicketFirst::Bónuszpontok::Bónuszszámla

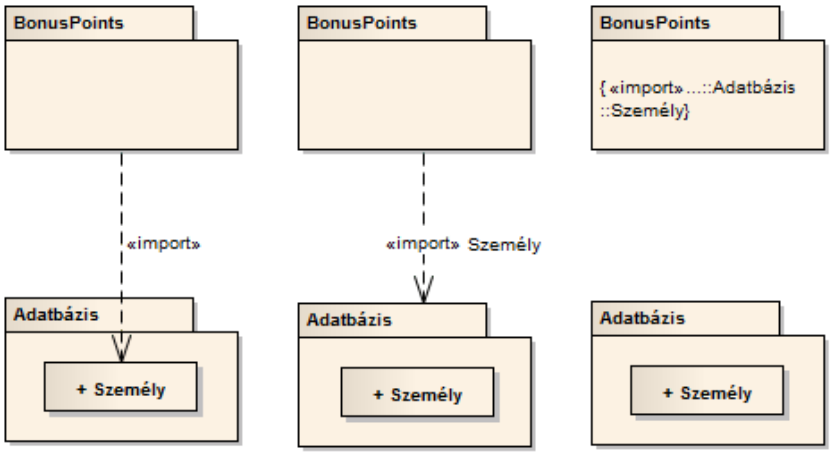
A csomagbeli elemekhez rendelhetünk láthatóságot, itt viszont az osztályok jellemzőivel ellentétben csak public és private értékek adhatóak meg. Ha nincs megadva láthatóság, akkor az elem nyilvánosnak számít. A nyilvános elemek minden névtérből elérhetőek a teljesen minősített nevük segítségével. Azonban az előző példából is látható, hogy a minősített nevek használata nem egyszerű. Mivel a nevek hosszúak, még egy ilyen egyszerű rendszernél is, és a rendszerek

átszervezésnél az ilyen hivatkozásokat is módosítani kellene, lehetőséget nyújt az UML egyfajta relatív útvonal megadására, amellyel hivatkozni lehet a különböző elemekre. Ezt az UML-ben bizonyos függőségek használatával érhetjük el, amelyet importkapcsolatoknak neveznek. Az importkapcsolatot szaggatott vonalú nyíllal jelöljük és az importáló csomag felöl az exportáló csomag felé mutat, valamint a nyílon látható a függőség fajtája. Megkülönböztetünk egyedi importot és csomagimportot.



10.3. ábra: A Bónuszpontok importálja a Személyt az Adatbázisból és átnevezi Nézőre

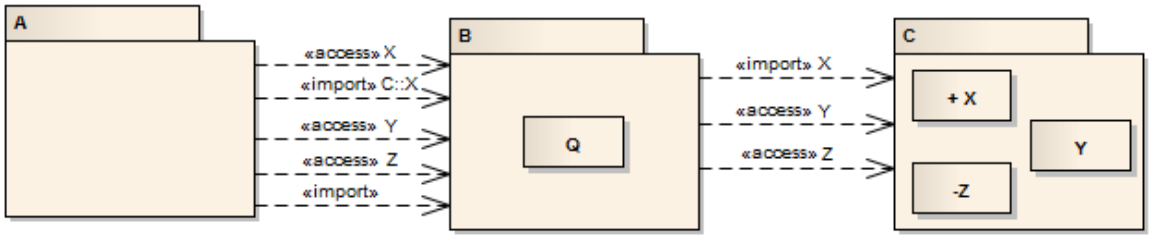
Az egyedi import a csomag elemeit külön-külön importálja. Három különböző jelölést lehet alkalmazni, amely a 10.4. ábra látható.



10.4. ábra: Egyedi import jelölésmódjai

Az első jelöléstípussal magára az importálandó elemre mutatunk. A második jelöléstípussal az exportáló csomagra mutat a nyíl, és a nyíl mellett jelenik meg az importálandó elem neve. A harmadik jelöléstípus esetén az importáló csomagon belül jelenik meg kikötéséként az import ténye.

Az egyedi import lehet nyilvános vagy privát. Nyilvános import esetén a nyíl az „<<import>> Elemnév” feliratot kapja meg. Privát, vagyis nem nyilvános import esetén a nyíl az „<<access>> Elemnév” feliratot viseli. Ez alapján meghatározható az importált elem láthatósága is, hiszen ezt a tulajdonságot az import tulajdonsága határozza meg, kivéve ha az importálandó elemre van az exportálandó csoporton belül láthatóság megadva. A 10.5. ábra és a 10.1-es táblázat mutat példát az importálandó elemek láthatóságára.



10.5. ábra: Importálás csomagok között

10.1. táblázat: A felsorolt csomagokból importálandó elemek láthatósága

Csomag	Elem	Láthatóság
A	X	Nyilvános
A	C::X	Nyilvános
A	Y	Nem látható
A	Q	Nyilvános
B	Q	Nyilvános
B	X	Nyilvános
B	Z	Nem látható
C	Y	Nyilvános
C	Z	Privát

Az importált elemet az import során át is lehet nevezni. Erre példa a Személy mint Néző importálása az adatbázisból.

Az egyedi elemek mellett a csomagok importálására is lehetőségünk van. Ilyenkor a csomag összes elemét importáljuk, átnevezés nélkül. Az importkapcsolatok grammatikája a következő:

Egyediimport ::= [element] Importfajta [Elem [as Újnév]]

Importfajta ::= <<import>>|<<element import>>|<<access>>|<<element access>>

Csomagimport ::= ImportFajta MinősítettNév

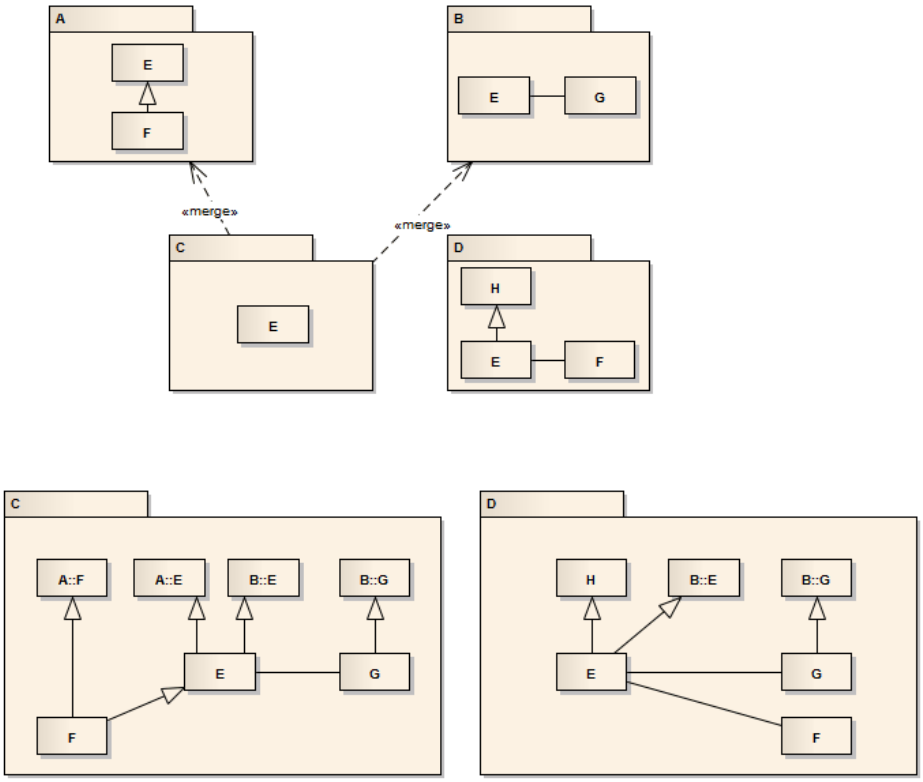
Csomaglláthatóság ::= [public|private]

ImportKikötés ::= {(CsomagImport|EgyediImport)}

MinősítettNév ::= Csomag::(Csomag)*Elem

Az előzőek alapján minden névtér két szegmensre osztható: a saját nevekre és az importált nevekre. A csomagon kívülről csak a nyilvánosként importált elemek láthatók. Az importálás tulajdonképpen egy hivatkozást jelent, tehát ha az importáló csomag törlődik, az exportáló csomag érintetlen marad.

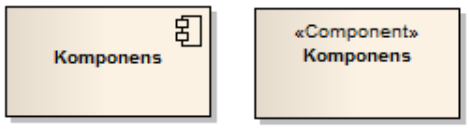
Egy csomag átfogó transzformációját a csomagbeolvasztás biztosítja. Az eredménye olyan összetett kapcsolatok kialakulása lesz, amelyben nem csak a névtér lesz hozzáférhető. A beolvasztási kapcsolatot szaggatott vonallal rajzolt nyitott hegyű nyíllal jelöljük és a nyílon a <<merge>> felirat található.



10.6. ábra: Csomagok beolvasztása és a beolvasztás eredménye

Komponensdiagram

Az UML-ben egy komponens egy egységbezárt, önálló, teljes és ezáltal cserélhető egység, amely függetlenül működtethető, telepíthető és összekapcsolható más komponensekkel. Ahhoz hogy egy komponens cserélhető legyen teljesen meg kell határozni a kontextusát és az interakcióit. A komponens az UML metamodellben az osztályok alosztálya, így rendelkezik azok összes jellemzőjével. Ennek következménye, hogy osztályokként ábrázolhatóak. A megkülönböztetés érdekében egy <<Component>> felirattal kell ellátni őket, vagy egy a komponensekre korábban használt szimbólumot is fel lehet tüntetni a jobb felső sarokban.



10.7. ábra: Komponensek jelölésmódjai

A komponenseknek lehetnek csatlakozóik, amiket interfészekkel írhatunk le. Ezekkel kapcsolódnak a kontextushoz.

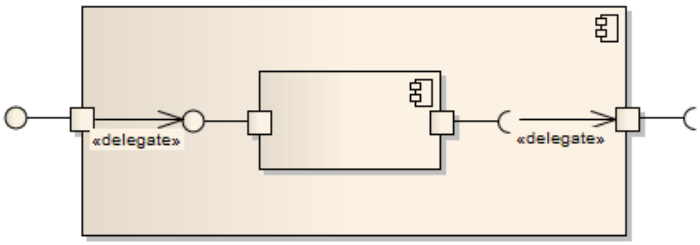
A komponenseknek többféle fajtáját definiálták. Ilyenek például a futási idejű komponensek, amelyek funkcionálisan kicsi, futási időben létező, a szakterülethez nem kötődő egységek. Speciális kibővítésekkel a használt programozási nyelv osztályaivá lehet átalakítani őket. A következő komponenstípus a tervezési komponensek, amelyek a szoftverfejlesztés nagyobb

építőkövei. Feladatuk a komponens nézőpontjainak összekapcsolása, legyen az a specifikáció, futtatható kód, vagy telepítési információ. A harmadik típus a megvalósítási komponensek típusa, amely a megvalósítás nézőpontjára vonatkozó komponens lesz. Az alrendszerek sztereotípiával rendelkező komponensek nagyméretű szakmai egységeket jelentenek, mint a TicketFirst szakarchitektúrájának a részei.

A komponensek közötti összekötőket meg kell különböztetni az alapján, hogy az összeköttetés egyenrangú komponensek között van e jelen vagy alárendeltségi viszonyban áll egyik komponens a másikkal. Az egyenrangú komponensek közötti összekötőket montázsösszekötőknek nevezzük. A montázsösszekötő egy nyújtott és egy igénybe vett szolgáltatást jelöl, jelölése a már ismert gömbcsukló jelöléssel történik (10.8. ábra).

10.8. ábra: Egyenrangú komponensek közötti összeköttetés

Ha a komponens és a részei – amelyek lehetnek szintén komponensek vagy osztályok – közötti összeköttetésről beszélünk, akkor a delegáló összekötőt alkalmazzunk. Ez a komponens hívásait továbbítja a megfelelő komponens számára, illetve lehetővé teszi a részek számára, hogy a hívásaik az őskomponens hívásaiként jelenjenek meg a kontextusban. Ha egy hívást egy olyan rész dolgoz fel, amely osztály, akkor a csatlakozó illetve a nyújtott interfész és az osztály között megvalósítási kapcsolat áll fenn (10.9. ábra).



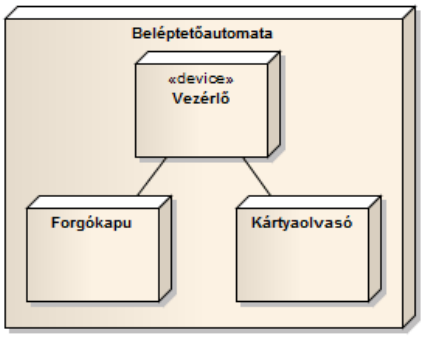
10.9. ábra: Delegáló összekötő nem egyenrangú komponensek esetén

Telepítési diagramok

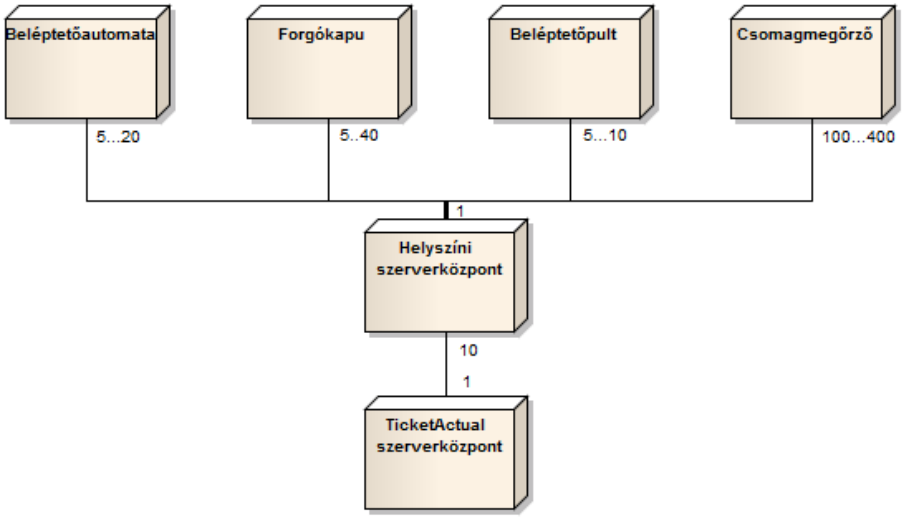
Egy rendszer megtervezésekor nem csak a szoftverrendszer megtervezésére kell gondolnunk, hanem arra a hardverre vagy eszközrendszerre is amelyen a program futni fog, illetve kapcsolatban lesz. Ez nagyban meghatározza a szoftver struktúráját, felépítését is, hiszen egy adatbázisszerver és egy alkalmazásszerver akár két külön hardveren is futhat. Előfordulhat, hogy más kapcsolatokat kell kialakítani ennek lekezelésére. Az UML a szoftver és a hardver összerendeléséhez a telepítési diagramokat nyújtja segítségül. Három változata ismert, amelyek szintén biztosítják az UML-ben már megszokott különböző nézőpontokat: a hangsúly egyiknél a rendszerstruktúrán, másiknál a szoftver rendszerstruktúrán történő kihelyezésén, a harmadiknál a telepítés részletein van.

Rendszerstruktúra diagram

A rendszerstruktúra diagrammal (10.10. ábra) egy rendszer számítási egységei és a köztük lévő kapcsolatok specifikálhatók. A struktúrában csomópontokkal találkozhatunk, amelyek élekkel vannak összekötve. A csomópontok egymásba ágyazhatóak, így taxonómiát is lehet hozzájuk készíteni. Lehet köztük öröklődési kapcsolatot is felállítani. A csomópontok altípusai az eszközök, amelyek fizikai erőforrásokat jelentenek, mint például az ábrán a Vezérlő, a <<device>> sztereotípiával ellátva.

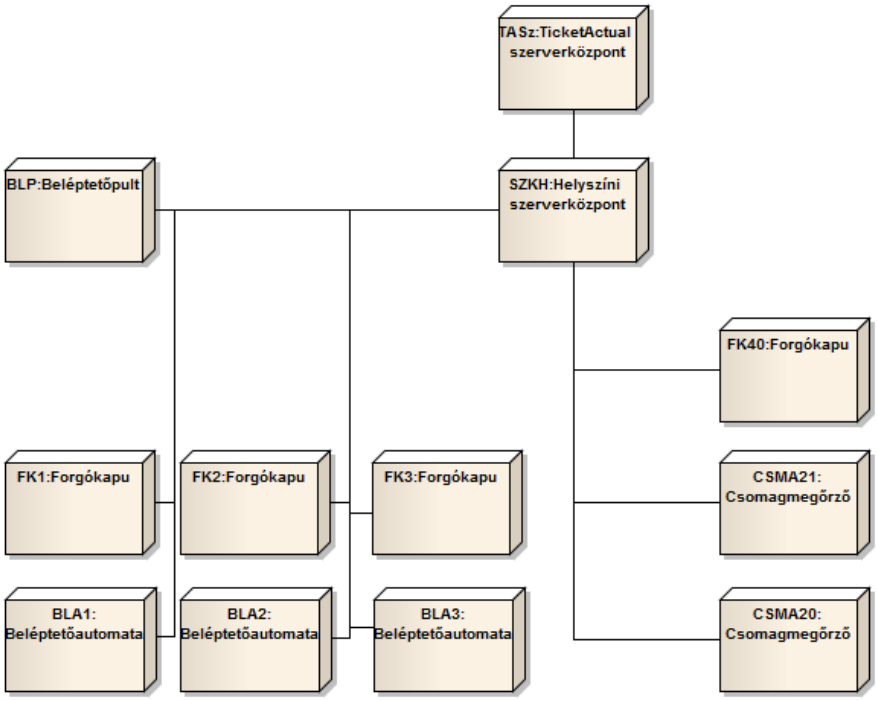


10.10. ábra: Csomópontok a rendszerstruktúra diagramon



10.11. ábra: A TicketActual rendszerének rendszerarchitektúrája

Az ábrázolás a rendszer architektúradiagramján (10.11. ábra) meglehetősen elnagyolt, további részleteket kell feltüntetni a diagramon ahhoz, hogy tényleges specifikációként lehessen alkalmazni. Ilyen például a tényleges csomópontok és az élek fizikai helyének meghatározása (10.12. ábra).

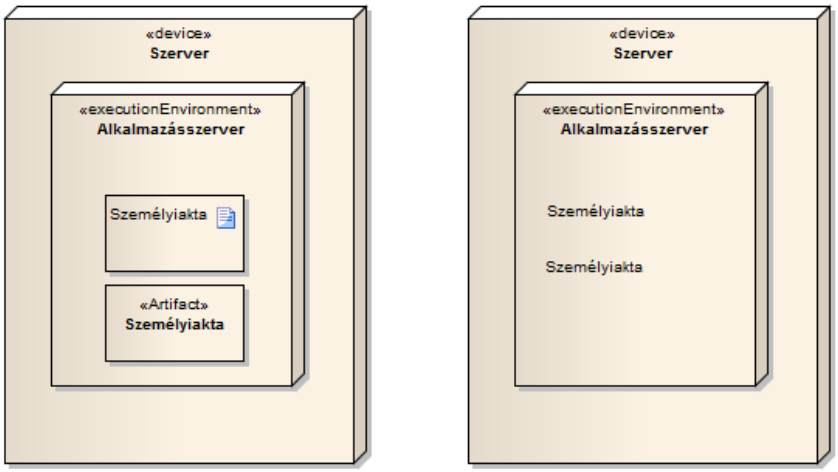


10.12. ábra: A rendszerstruktúra egy kisebb beléptetőkapunál

Az architektúra egyes részletei is tovább finomíthatóak: a rendszerközpontban lévő architektúrát fel lehet bontani szerverekre, köztük lévő kapcsolatot biztosító hálózati eszközökre.

Kihelyezési diagram

A kihelyezési diagram annyiban különbözik a rendszerstruktúra diagramtól, hogy a rendszerben található számítási egységeken belül az úgynevezett köztes termékeket is elhelyezzük. Ilyenek lehetnek a különféle dokumentumok, állományok, adatbázistáblák, email-ek… Ezek a termékek kihelyezhetőek a csomópontokra. Ennek megfelelően rendelkeznek egy „element” és egy „location” attribútummal, amely megadja az elemet és az elhelyezés helyét. A termékeket téglalappal ábrázoljuk, és az <<artifact>> sztereotípiával vagy egy dokumentum szimbólummal a jobb felső sarokban azonosítjuk. A kihelyezéssel a köztes terméket egy csomóponthoz rendelhetjük. Ennek jelölésére a termék csomópontra helyezése illetve a termékek listájának csomópontra írása szolgál (10.13. ábra).



10.13. ábra: A köztes termékek csomópontra helyezése

A kihelyezési diagram kapcsán meg kell említenünk még egy speciális csomópontot, amit <<execution environment>> sztereotípiával jelölünk. Ez a futási környezet, amely biztosítja a rendszernek azt a háttér-szoftverrendszert, amelyen működni képes.

11. Az interakciók modellezése

Ebben a fejezetben visszatérünk a folyamatközpontú modellezéshez. Az interakciók tulajdonképpeni üzenetváltások különböző résztvevők között. Ezek a résztvevők lehetnek osztályok, aktorok, komponensek, csatlakozók vagy csomópontok. Az interakciós diagramok alkalmasak a rendszerek folyamatainak leírására, dokumentálására, követelmények vagy tesztesetek specifikálására.

Az interakciós diagramoknak három fő típusát találjuk mag 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.

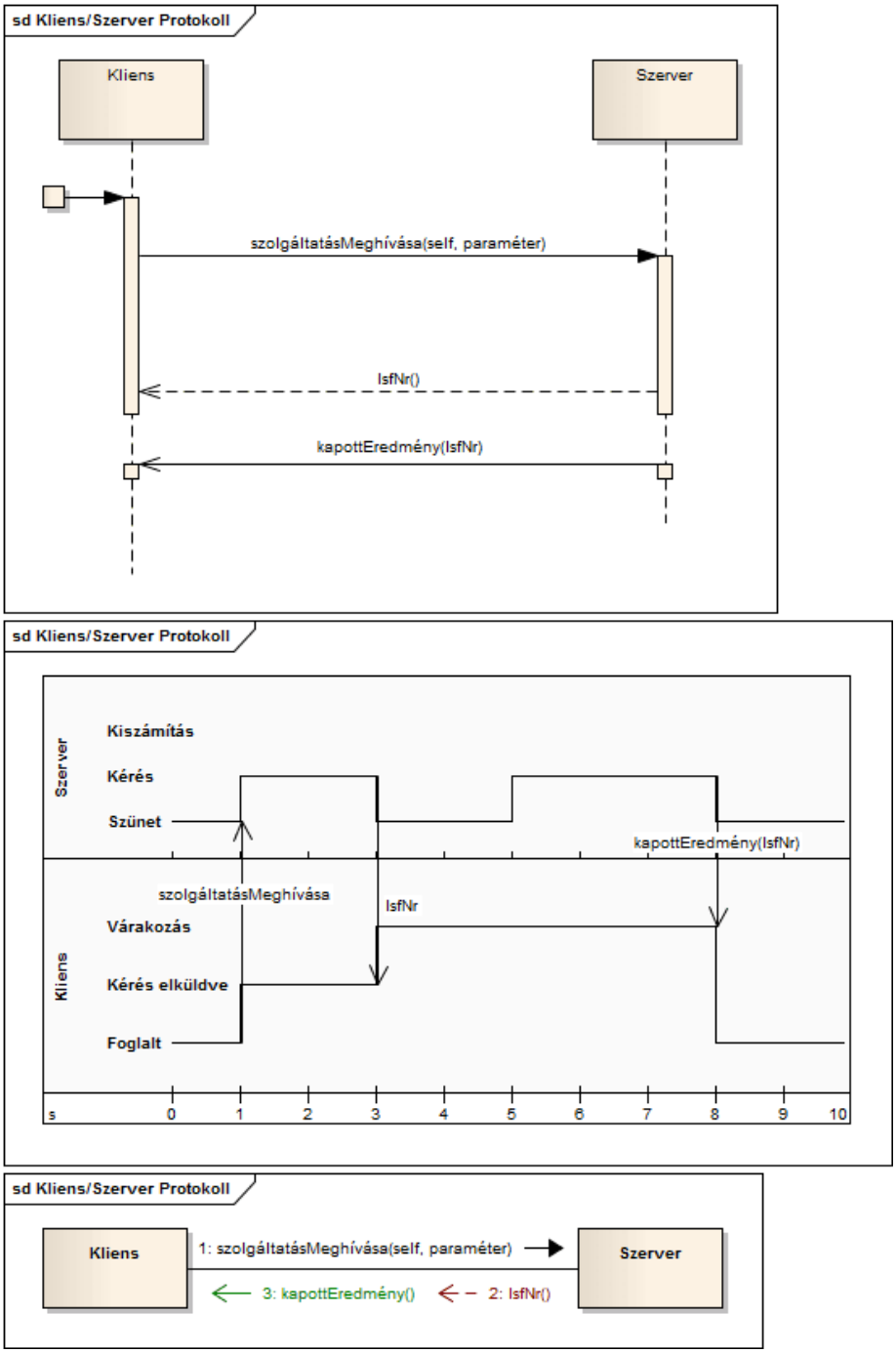
Osztályok közötti interakciók: szekvenciadiagramok, idődiagramok, kommunikációs diagramok

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 (11.1. ábra).

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.



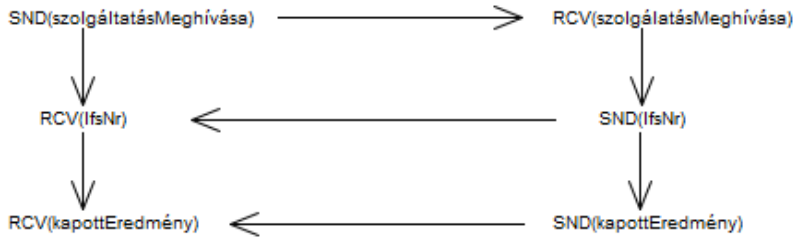
11.1. ábra: Kliens/szerver kölcsönhatások szekvenciadiagramon, időDiagramon és kommunikációs diagramon

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

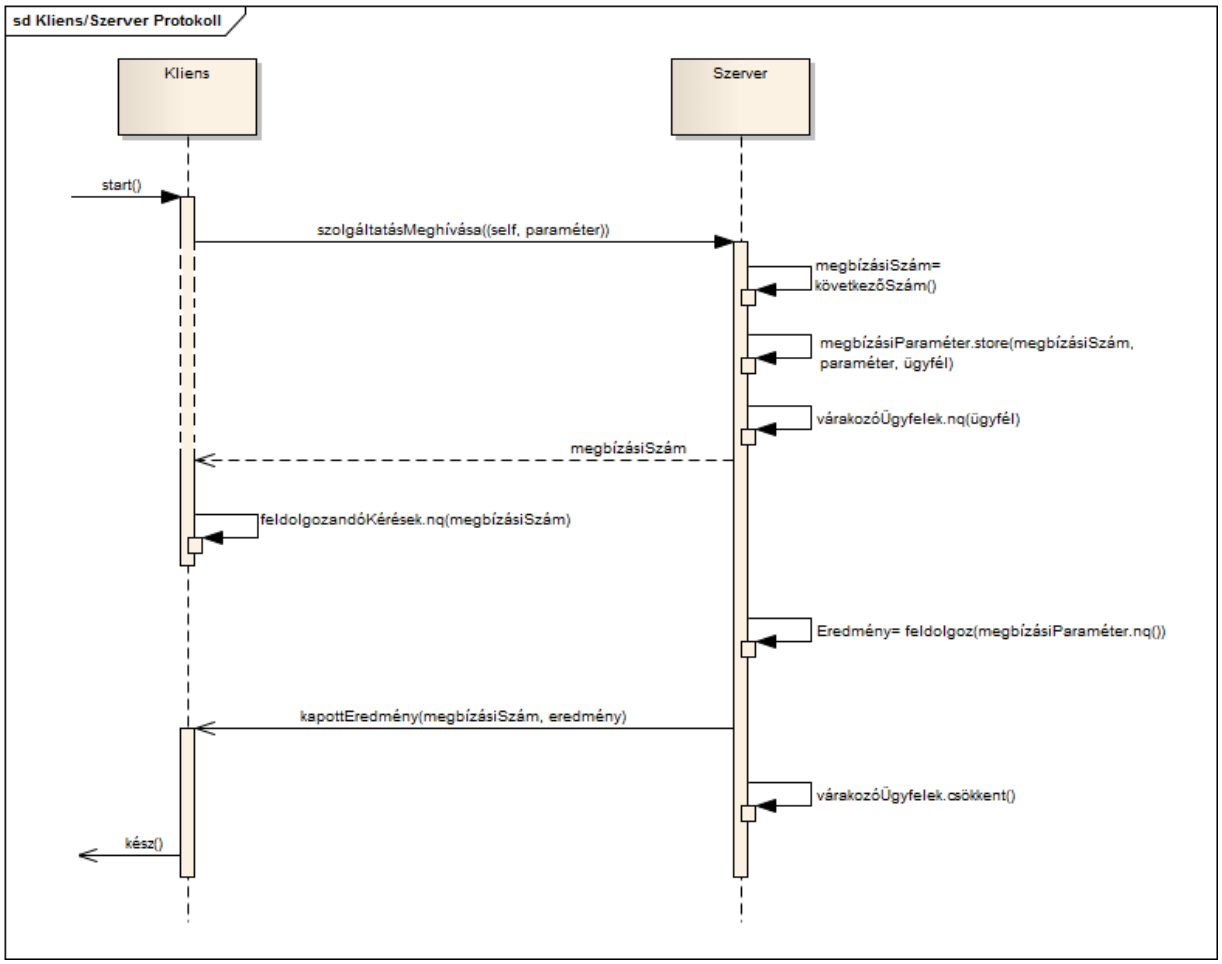
Az interakció interakciós események egymásutániságának folyamata. Az események egymásutániságának sorrendjét két szabály határozza meg: az egyik, hogy az üzenet küldése mindig megelőzi az üzenet fogadását, a másik pedig, hogy az esemény előfordulások sorrendjét az életvonalon való megjelenésük sorrendje határozza meg. Ennek a két szabálynak a leírása alapján a 11.2. ábra mutatja az előző ábrákon látott kliens/ szerver kapcsolat eseményeinek egymás-utániságát. Természetesen ebből nem tudunk meg semmit a pontos időbeni távolságokról.



11.2. ábra: Az esemény-előfordulások egymásutániságának összefüggései

Már volt szó róla, hogy 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 (11.3. ábra) 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.



11.3. 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

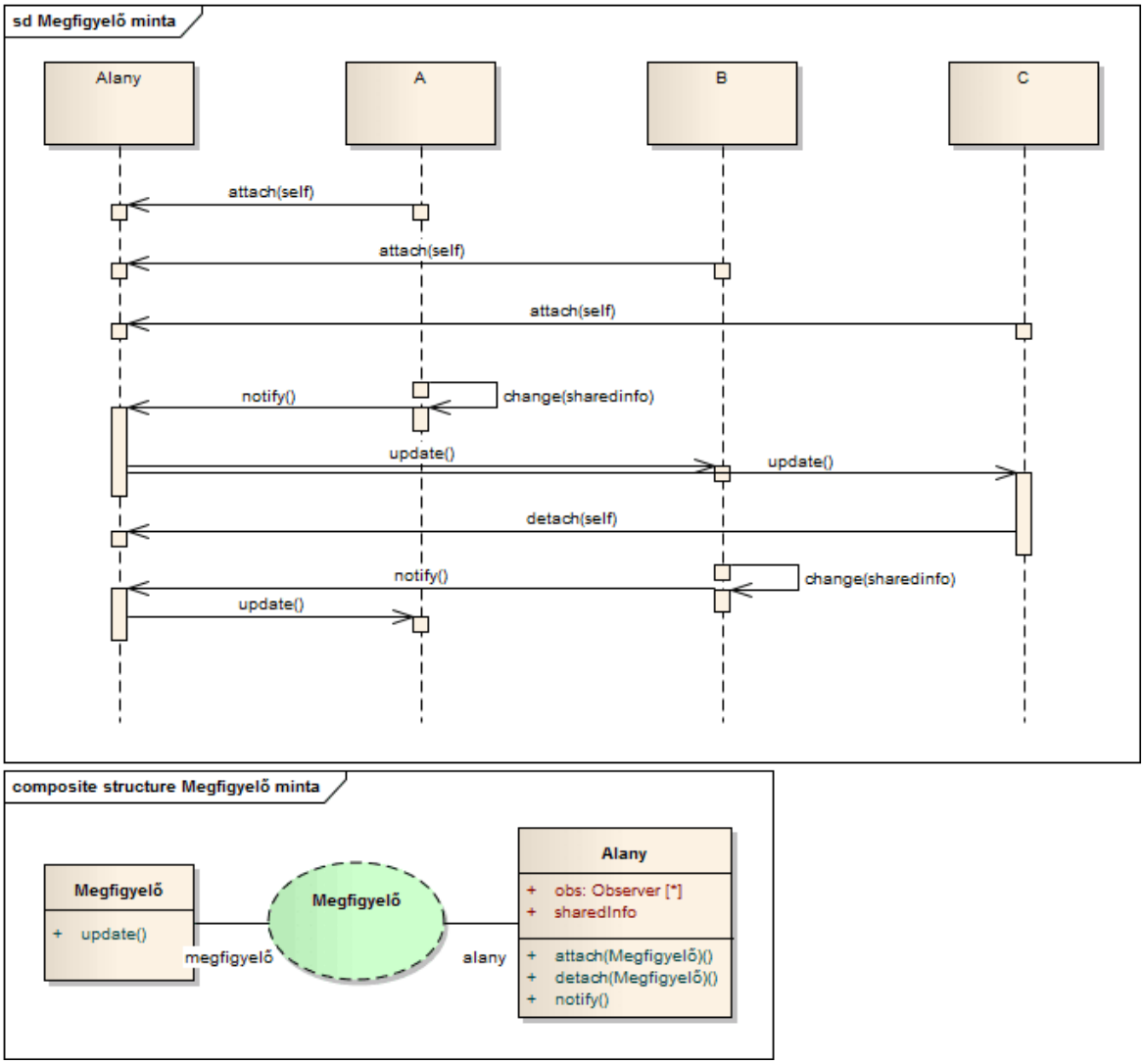
Argumentumok ::= (Argumentum (,Argumentum)*)

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.

A 11.4. ábra 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.

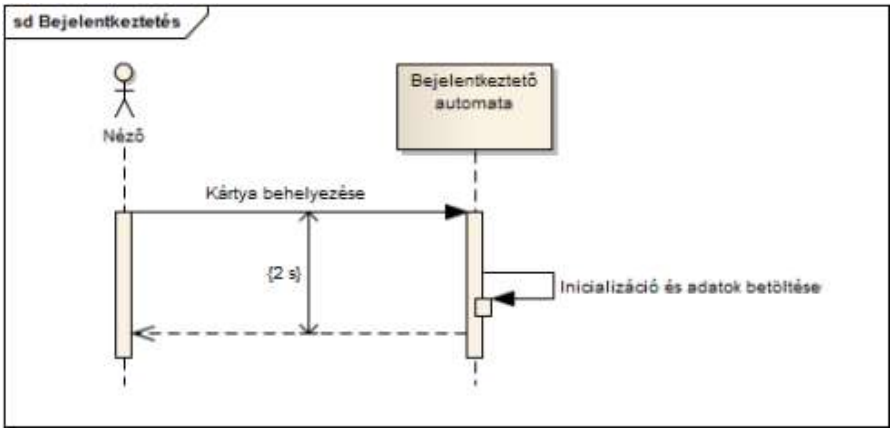


11.4. ábra: A megfigyelő tervezési minta interakciós diagramja és struktúrája[7]

Kontextusinterakciók

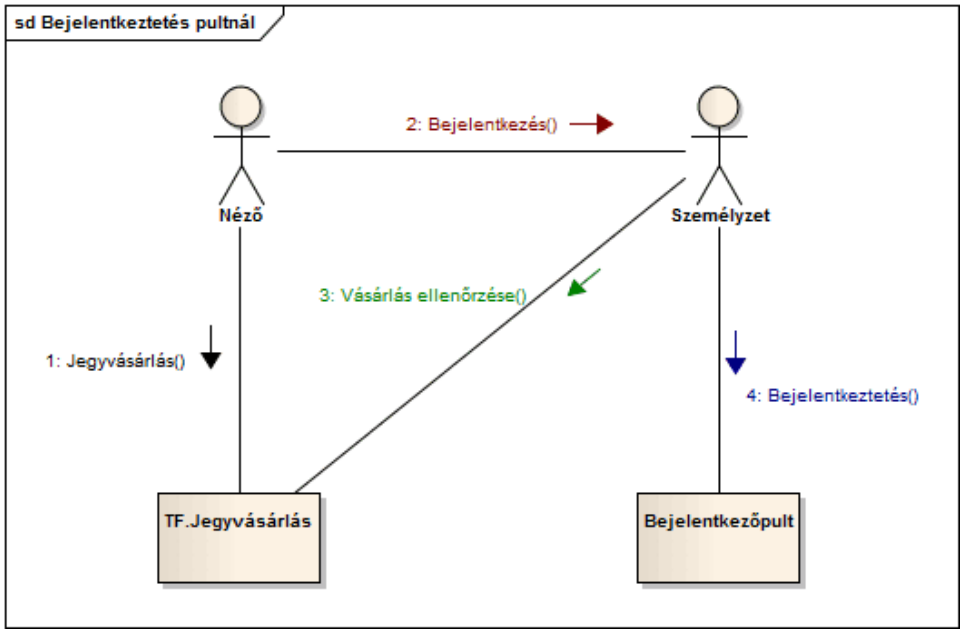
A kontextus-együttműködés a rendszer és a környezete közötti együttműködést mutatja be. A dinamikus viselkedését kontextusinterakciókkal lehet modellezni. A 11.5. ábra látható, hogy a Néző behelyezi a kártyát a Beléptetőautomatába. Az automata inicializálja magát és rákeres a Néző adataira, majd üdvözli és megmutatja az ülőhelyét a helyszín térképén.

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



11.5. ábra: Kontextusinterakció

Egy újabb példa látható az alábbi ábrán (11.6. ábra):



11.6. ábra: Interakció több partner esetén

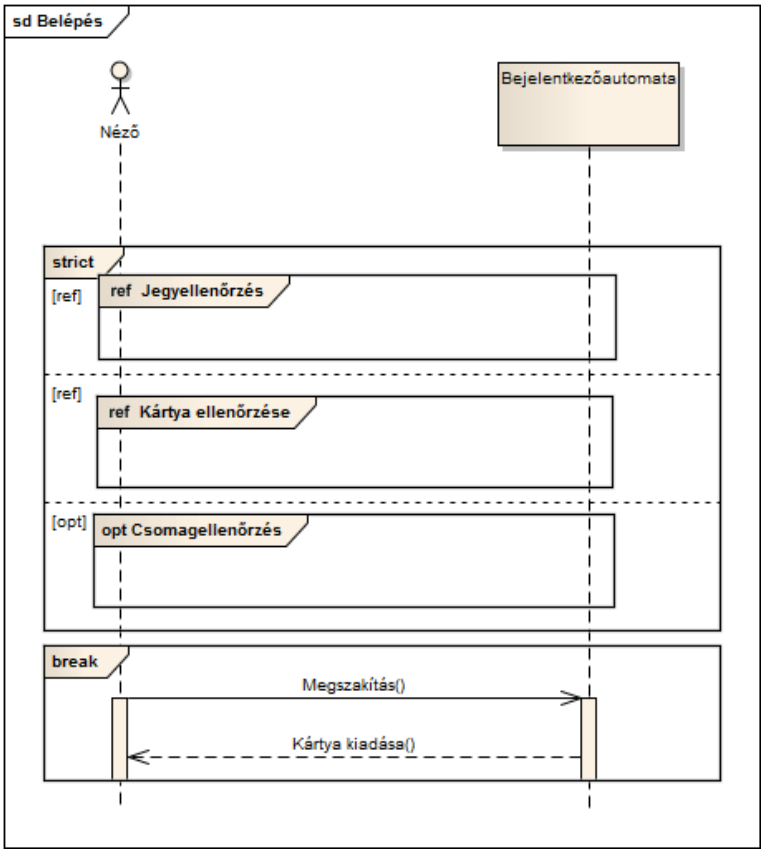
Az eset akkor következik be amikor a kártyás azonosítással valamilyen gond van és a Személyzetnek kell a Néző segítségére sietnie. Saját képernyőjén kártya számának beírásával vagy a Néző személyes adatai alapján elvégzi a jegyellenőrzést és a Néző azonosítását

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.

Interakciós operátorok

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 (11.7. ábra), amelyek segítségével több interakciót kombinálhatunk, így folyamatok egész halmazát írhatjuk le.



11.7. 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

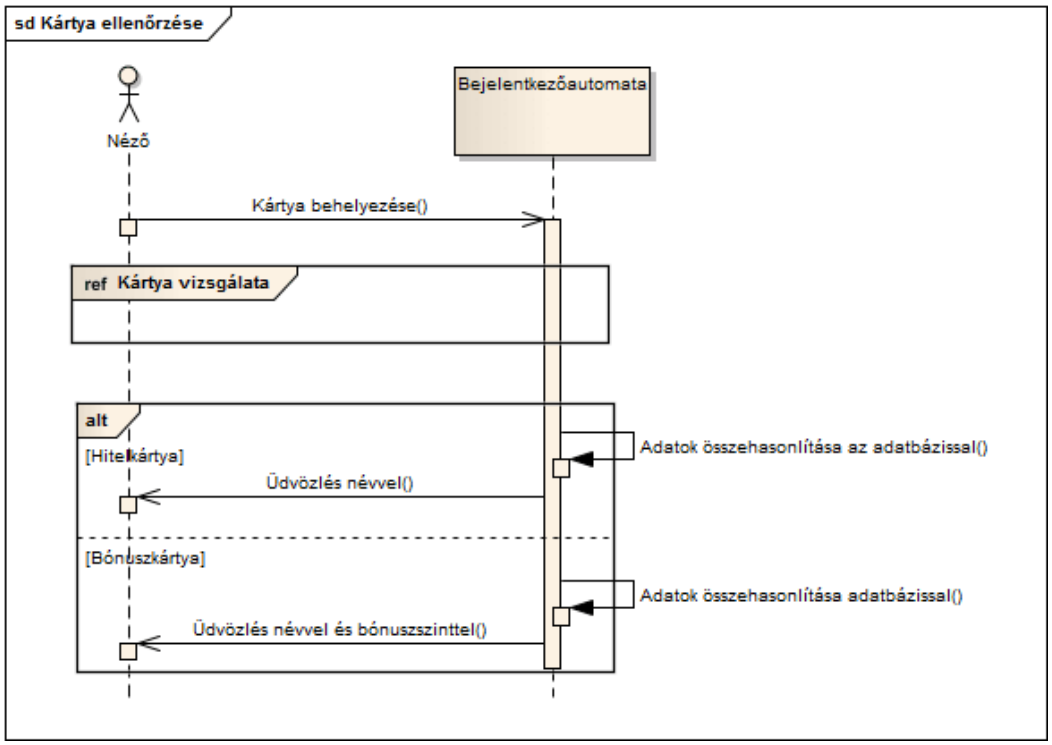
Az operátor más interakciókra való hivatkozást jelöl. Ezek az interakciók más diagramokon lehetnek kifejtve, de tartalmuk akár a strict közvetlen operandusaként is megjelenhetne.

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.

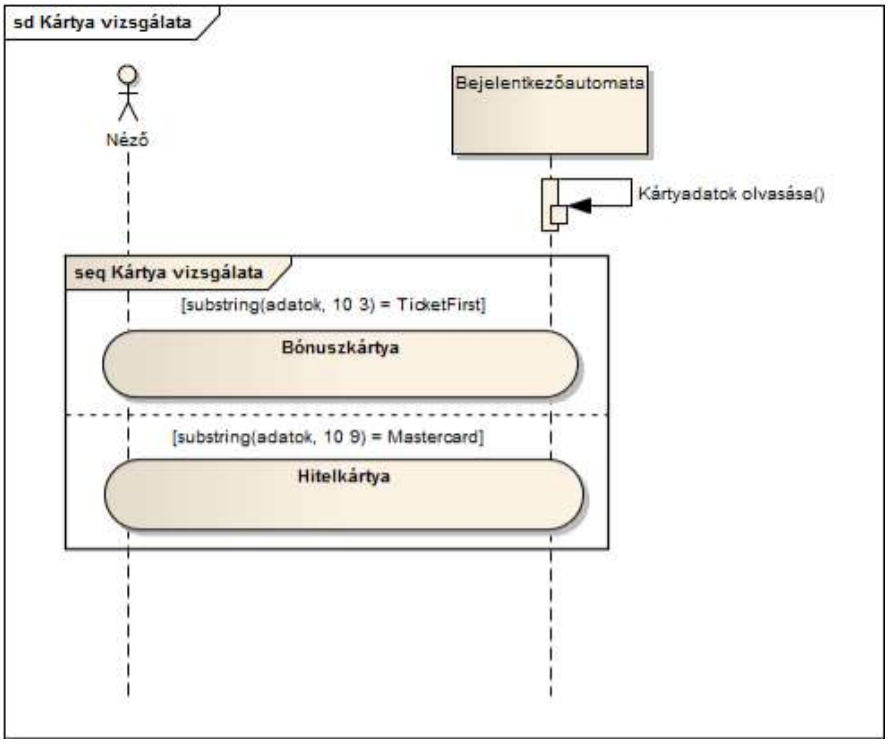
Alt operátor

Ha finomítjuk a Néző azonosítása operátort, akkor a különbséget kell tenni a hitelkártyával és a bónuszkártyával történő azonosítás között (11.8. ábra). Ezt az alt operátorral tehetjük meg. Az alt operátor jelen esetében két operandussal rendelkezik, amelyek alternatív szituációkat jelentenek.



11.8. 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 (11.9. ábra). A kártya vizsgálata interakción két ugrójelet, a Bónuszkártyát és a Hitelkártyát azonosíthatunk, amelyek a Néző azonosítása esetén esetválasztás feltételeiként kerülnek elő.



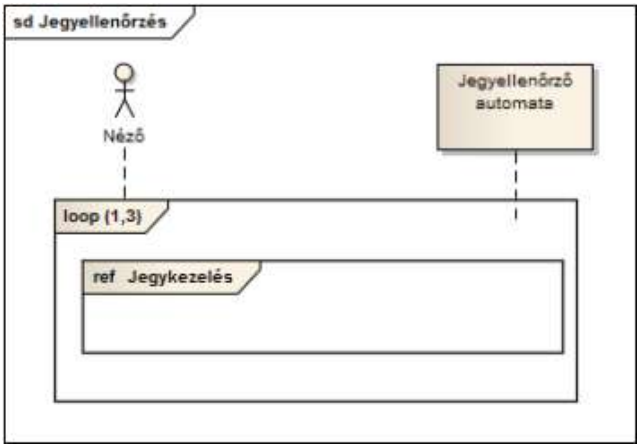
11.9. ábra: Ugrójelek használata

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.

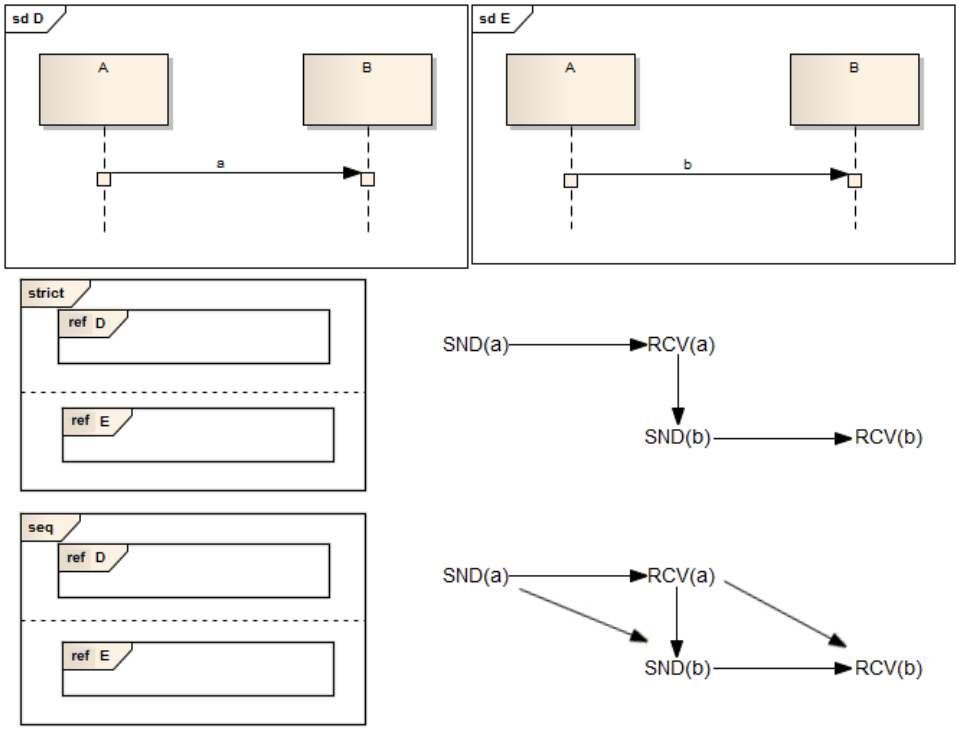
Seq és loop operátorok

A Nézõ több jegyet is kezeltethet az automatánál. Ennek a megszorításnak a kezelésére alkalmas a loop operátor (11.10. ábra). 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.



11.10. 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. A 11.11. ábra erre láthatunk példát.

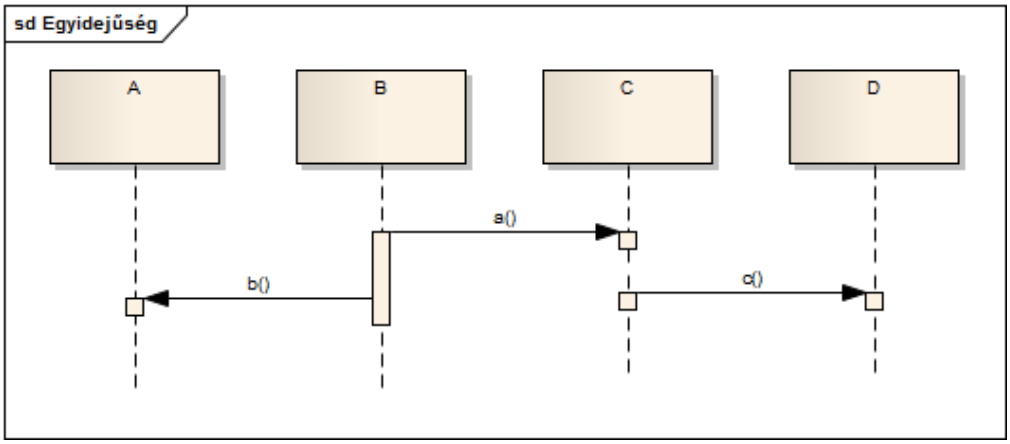


11.11. ábra: A seq és a strict operációk közötti különbség

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

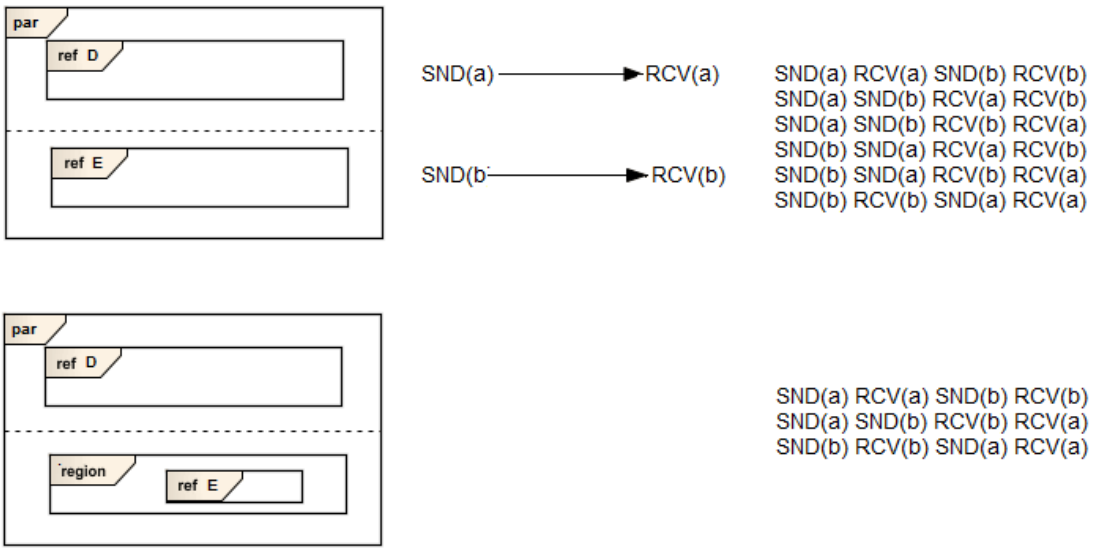
Az egyidejűséget az interakciós diagramról nem lehet egyértelműen leolvasni, mivel az interakciós partnerek köztes időben fellépő állapotairól és egyéb interakcióiról nem esik szó. Így például nem lehet egyértelműen eldönteni a 11.12. ábra, hogy elsőnek a b vagy a c üzenet kerül korábban elküldésre.



11.12. 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 (11.13. ábra).



11.13. 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 (11.13. ábra).

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

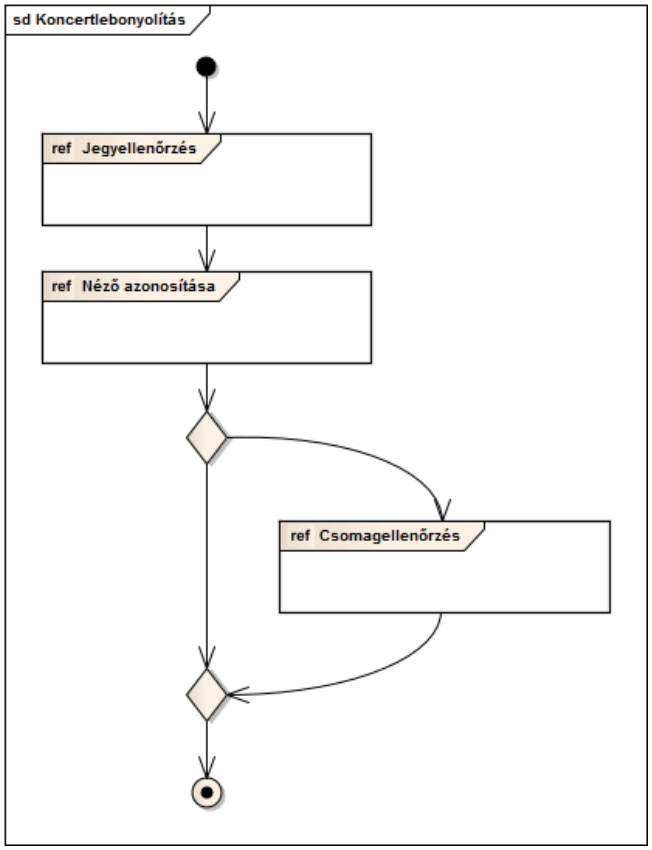
consider(P,Z): az ignore ellentettje, csupán a Z-beli üzeneteket veszi figyelembe az interakció során

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. A 11.14. ábra látunk erre példát.



11.14. ábra: Belépés megjelenítése interakciós áttekintés 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.

12. Kivételek kifejezése - OCL: Object Constraint Language

Az előző fejezetekben megismerhettük az UML strukturális és viselkedésdiagramjait, láttuk milyen lehetőséget nyújt egy rendszer modellezésére. Bizonyos megszorítások csak nagyon nehézkesen adhatóak meg ezeken a diagramokon. Egyes helyeken láthatunk természetes nyelven leírt feltételeket természetes nyelven megfogalmazva, megjegyzésdobozokban. Ilyen megszorítások lehetnek a futási időre vonatkozóak, elő- és utófeltételek, előfordulások számának meghatározására vonatkozó előírások.

Az UML a megszorítások formális leírására az OCL-t javasolja. Az OCL más néven Object Constraint Language, egy olyan nyelv, amelyen egyszerű megszorításokat, invariánsokat, valamint elő- és utófeltételeket lehet definiálni. Az OCL valójában nem az UML része, az OMG részéről egy ajánlás, ez alkalmazható a megszorítások megadására. Bármely más programozási nyelv alkalmas lenne erre, de használatuk inkább az implementációs fázisban javasolt. Az OCL akkor használható jól, ha egy természetes nyelv túl informális a modellezési környezetben.

Az OCL a legtöbb diagramtípusnál használható lenne, mégis leginkább az osztálydiagramokra korlátozódik az alkalmazása.

Az OCL típusai

Az OCL egy erősen típusos nyelv, minden OCL kifejezés típussal rendelkezik[8]. A nyelv típusainak leírásról egy hasznos összefoglaló anyag található az alábbi weboldalon: http://fondement.free.fr/objx/netsilon/xion/predefined/overview-tree.html.

A nyelv előre definiált típusokkal dolgozik, ezek minden kifejezésben elérhetőek. A műveleteket csak azonos vagy egymásból származtatott típusokon lehet végezni. Nem lehet például egy szöveg típusú változóhoz egy egész számot hozzáadni. Az OCL alaptípusai és a rajtuk alkalmazható műveletek a következők (12.1. táblázat):

12.1. táblázat: Az OCL alaptípusai

Típus	Példa	Műveletek
boolean	true, false	and, or, not, implies, if-then-else
integer	1, 0, -3, 642	+, -, *, /, abs(), <,<=, >, >=, =, <>, …
real	1, 4, 4.56	+, -, *, /, floor(), <,<=, >, >=, =, <>, …
string	'TicketActual'	contact(), size(), substring(), …

Ezeken kívül az OCL számos más típussal rendelkezik. Az OCL-ben definiált típusok az OclType egy példányának tekinthetőek. A típus bizonyos kapcsolatot jelent a metamodellhez. Az OclType segítségével a következő információk határozhatóak meg: a típus neve, a típus attribútumainak,

[8] http://www.omg.org/spec/OCL/2.2/PDF

asszociáció végpontjainak, a típuson értelmezett műveleteknek a nevei, a típus közvetlen illetve összes őstípusa, a típus összes példánya.

Az OclAny az OCL összes beépített alaptípusának valamint a modellben definiált összes típusnak őstípusa. Nem őse viszont a kollekció típusoknak. Összes művelete elérhető bármely objektumpéldányon keresztül. Ezek a következők: egyenlőségvizsgálat, leszármazás vizsgálata (oclIsKindOf(), oclIsTypeOf()), explicit futásidejű típuskonverzió (casting, oclAsType()). Csak olyan típusra lehet érvényesen konvertálni, amely a konvertált példány típusának őstípusa.

Az oclIsInState() művelet paramétere, példányai egy állapotgép állapotainak felelnek meg. Az oclInState() lekérdezi, hogy egy objektum az állapotgép adott állapotában tartózkodik-e.

Minden OCL kifejezés egyben az OclExpression osztály egy példánya is. Az OclExpression a kifejezések szemantikai értékének meghatározásakor kap szerepet, valamint egyes gyűjteménytípusokon értelmezett művelet paramétere lehet.

Modell típusok

A modellben szereplő összes osztály az OCL nyelvben típusként jelenik meg. Az UML modell minden olyan osztályának, amely szerepel OCL kifejezésekben, OclAny az ősosztálya. A modellben definiált osztályok példányainak attribútumai, valamint azon műveletek, amelyek az objektumok állapotát csak olvassák, de nem módosítják, elérhetők az OCL kifejezésekből. Az osztályok közötti asszociációk mentén navigáció azonosítható.

Kollekciótípusok

Az absztrakt Collection osztály, és a belőle származó Bag, Sequence és Set osztálysablonok az OCL nyelv egyedüli elemeiként nem az OclAny leszármazottai (12.1. táblázat). A Collection osztály absztrakt ősosztály, az összes kollekcióra jellemző művelettel: méret lekérdezés, ürességvizsgálat, elemek tartalmazásának és előfordulásának vizsgálata, rendezett Collection, Sequence létrehozása a gyűjtemény elemeiből. A rendezéshez megadható egy kifejezés, melynek eredménye lesz a rendezés kulcsa. Összegzés, eredménye az összes gyűjteményben levő elem összege a rajtuk definiált + művelet segítségével.

12.2. táblázat: Az OCL-ben definiált kollekciótípusok

Típus	Példa	Műveletek
set	set{}, set{8, 5, 2}	asSequence (), size(), union(), intersection(),…
orderedSet	orderedSet {8, 5, 2}	asSet (), size(), union(), intersection(),…
bag	bag {3, 3, 4, 5, 5, 6}	asSet (), size(), union(), intersection(),…
sequence	sequence {1, 2, 3}	asBag (), size(), union(), intersection(),…

A Set matematikai értelemben vett halmaz, amelyben egy elem csak egyszer szerepelhet. Nincs rendezés megadva a benne lévő elemek között.

A Bag olyan gyűjtemény, amelyben a Set-el ellenétben egy elem többször is előfordulhat. Az elemek között itt sincs sorrend rögzítve. Műveletei megegyeznek a Set műveleteivel, de mindegyiknek van egy saját paraméterhez igazodó változata.

A Sequence olyan gyűjtemény, amelyben az elemek sorrendje rögzített. A sorozatban egy elem többször is előfordulhat. Műveletei: méret lekérdezés, összefűzés, elem hozzáadása a sorozat elejéhez vagy végéhez, adott pozícióban levő elem vagy részsorozat lekérdezése, iterációs műveletek (select(), reject(), collect(), iterate()).

Összetételtípus

Az összetett szerkezetű típus az összetételtípus (Tuple) (12.3. táblázat). Az összetétel típus a programozási nyelvekből jól ismert struktúratípusnak (struct a C/C++ nyelvekben) vagy rekordtípusnak (record a Pascal/Delphi nyelvekben) felel meg.

12.3. táblázat: Az OCL összetételtípusa

Típus	Példa	Műveletek
tuple	tuple { jegyszám = 214365, kezdés = 19:00}	-

Tulajdonságok

A modell objektumainak tulajdonságait az objektum neve és a tulajdonság neve közé írt '.' jellel érhetjük el. Az OCL-ben a tulajdonságok közé sorolhatóak az attribútumok, a műveletek, az asszociációk, a navigáció.

Az attribútumok jelentik a legegyszerűbb tulajdonságot, értékük megegyezik a vizsgált példány modellben definiált azonos nevű attribútumának értékével.

Minden OCL kifejezés egy adott környezetben érvényes, amit minden esetben meg kell adni kivéve, ha a megszorítás a diagramon jelölve van. Amennyiben nincs megadva a kifejezés környezete, a definiálása a context szóval lehetséges:

```
context TipusNev
```

Adattípustól függően különböző műveleteket lehet végezni az adott típusokon. A művelet megfeleltethető egy metódusnak, annak függvényében, hogy a leképezendő nyelvben létezik e a metódus által végrehajtandó operáció. Függvényhívás a -> karakterpárossal történik. Például: fizetes->sum(). Ez a metódus egy kollekciótípus metódusa, ami a tárolt elemek összegével tér vissza. Az alaptípusok végzett műveleteit egy pont segítségével lehet meghívni. Például: osszeg.round.

A műveleteknek lehetnek paramétereik, melyet a C típusú nyelvekhez hasonlóan zárójelek közé kell írni. A zárójelek megadása üres paraméterlista esetén is kötelező. A műveletekre az OCL

kétféle megszorítás megfogalmazását teszi lehetővé: ezek az elő- és utófeltételek. Az előfeltételekkel a függvény meghívása előtti feltételeknek kell teljesülnie, az utófeltételnél a végrehajtás utáni megszorítások definiálhatóak. Először a környezetet kell megadni, ami egy metódus visszatérési és paraméterértékekkel. A visszatérési értékre a result kulcsszó használatával lehet hivatkozni. Az előfeltétel a pre, az utófeltétel a post kulcsszavakkal történik:

```
context Konyv :: Oldalszam(osz:Integer) : Boolean
pre : osz > 1
post : result = true
```

Az OCL lehetőség ad az osztálydiagramon definiált asszociációk mentén történő navigációra. Ez tulajdonképpen megegyezik tulajdonságértékének lekérdezésével. A tulajdonság neve az asszociáció azon végének a neve, amely nem a vizsgált objektumhoz kapcsolódik. Ha az asszociáció szóban forgó végpontja névtelen, névként a végpontnál levő osztály kisbetűvel kezdődő nevét használhatjuk. Amennyiben ez többértelműséghez vezet, akkor az asszociációt kötelező megkülönböztető névvel ellátni. Ha az asszociációs kapcsolat számossága 0..1 vagy 1, a navigáció eredménye egy objektum. Egynél nagyobb számosságú asszociáció vég felé történő navigáció Set eredményt ad. Rendezett (ordered) jelöléssel ellátott asszociáció esetén az eredmény Sequence. Lehetőség van arra is, hogy az egy vagy nulla számosságú eredményt is halmazként kezeljük, ehhez az eredményül kapott objektumra, mint halmazra kell alkalmazni a halmazon definiált műveleteket a nyíl ('->') jelölés segítségével. Az OCL automatikusan egy- vagy nullaelemű halmazokra való hivatkozásként kezeli az ilyen hívásokat.

Az OCL szintaxis szerint az asszociációs osztály is tulajdonságként érhető el. A tulajdonság neve az asszociációs osztály kisbetűvel kezdődő neve lesz. Rekurzív asszociáció esetén, amikor az asszociáció mindkét végpontja ugyanarra az osztályra vonatkozik, minősíteni kell az osztályt a megfelelő végpont nevével. Az asszociációs osztályból történő navigáció esetén a szerepnevet kell tulajdonságnévként használni. A navigáció eredménye mindig egyetlen objektum, de itt is használható az implicit halmazként való kezelés a nyíl jel segítségével.

A minősített asszociációk esetén a tulajdonság választja ki, hogy az asszociáció egyik végpontjában szereplő számos példány közül melyikre vonatkozik a kifejezés. Ezen tulajdonság értékeit az asszociáció szerepneve után [] jelek közé zárva írjuk. Amennyiben a tulajdonságokat elhagyjuk, a navigáció eredménye az asszociáció végpontjában szereplő összes objektum.

Amennyiben egy osztály elfedi valamely ősosztályának egy műveletét, attribútumát vagy asszociációját, explicit típuskonverziót kell alkalmazni, hogy az eredeti tulajdonságot el lehessen érni. Ez az OclAny osztály oclAsType() műveletével tehető meg, amelynek paraméterként az ősosztály típusát kell átadni.

Kollekciókon definiált műveletek

A kollekciókon végezhető legfontosabb műveletek iteráló jellegűek, paramétereik között szerepel egy OCL kifejezés, melyet a gyűjtemény minden tagjára ki kell értékelni.

Ha egy kollekciónak csupán néhány elemére van szükség, akkor a select() művelet használható. Három formában hívható, ezek közül a legegyszerűbb paraméterként egy kifejezést vár, mely kiértékeléskor Boolean értéket ad eredményül. A kifejezés iteratív módon a gyűjtemény minden tagjának kontextusában lefut, amelyekre igaz eredményt ad, benne lesznek a select() által visszaadott kollekcióban.

A reject() azokat az elemeket adja eredményül, amelyekre a '|' jel után megadott feltétel nem teljesül.

A collect() művelet új kollekció létrehozására szolgál. Az új kollekciót úgy kapjuk meg, hogy kiindulási kollekció minden elemére kiértékelődik a collect() paraméterében átadott kifejezés, majd ennek értéke kerül az új kollekcióba.

A forAll() illetve exists() műveletek az előzőekben említett műveletekhez hasonlóan iterálnak az elemeken. A paraméterként kapott kifejezés Boolean típusal értékelődik ki. A forAll() művelet eredménye igaz, ha a kifejezés minden egyes elemére igaz értékkel értékelődött ki. Az exists() művelet akkor ad igazat eredményül, ha legalább egy olyan elem van a gyűjteményben, amelyre a kifejezést kiértékelve igazat kapunk eredményül.

Az iterate() a legáltalánosabb iteráló művelet, segítségével az előzőekben ismertetett műveletek mindegyike kifejezhető.

Precedencia szabályok

Az OCL nyelvben a precedencia szabályok a következők. Legelöl szerepel a legmagasabb precedenciájú művelet.

- '@pre', utófeltételben végrehajtás előtti állapotra hivatkozás
- '.' és '->', pont és nyíl, tulajdonsághívás
- unáris 'not' és '-'
- '*' és '/', szorzás és osztás aritmetikai operátorok
- '+' és bináris '-' aritmetikai operátorok
- 'if-then-else-endif'
- '<', '>', '<=' és '>=' relációs operátorok
- '=' és '<>' relációs operátorok
- 'and', 'or', 'xor' logikai operátorok
- 'implies' logikai operátor

Az operátorok precedenciája a szabvány részét képező nyelvtanban rögzítve van, tehát az elemzés során nem kell rá külön gondot fordítani.

Invariánsok

Az invariánsok az Osztályokra vonatkozó megszorítások, minden példányára érvényes. Nem létezhet olyan objektum, ami a megadott invariánsnak nem tesz eleget. Ha például egy Könyv

osztályra invariáns típusú megszorítás érvényes, akkor a Könyv osztály összes példányára teljesülnie kell a kifejezésben leírtaknak. Megadása a következő kifejezéssel lehetséges:

```
context Konyv inv : self.Oldalszam >= 1
```

A self kulcsszóval magára az osztályra való hivatkozás valósítható meg. A következő kifejezés ugyanezt jelenti:

```
context k : Konyv inv : k.Oldalszam >= 1
```

Az invariáns megszorításoknak nevet is lehet adni:

```
context k : Konyv inv MinimalisOldalSzam : k.Oldalszam >= 1
```

Csomagok

Több megszorítás egységes kezelésére, egy állományban való leírására csomagok használata szükséges. Megadása a package és az endpackage kulcsszavakkal történik:

```
package CsomagNeve::AlcsomagNeve
context A inv:
…
context A::muveletNeve( … ):Tipus
pre: …
endpackage
```

Kezdeti- és származtatott értékek

A kezdeti érték nem más, mint az attribútum első értéke, vagyis az osztály példányosításakor felvett érték. A származtatott érték esetén a kifejezésben leírtak szerint változhat az értéke. A kezdeti az „init”, a származtatott érték a „derive” kulcsszavakkal adható meg:

```
context Programozo::Statusz:String
--Kezdeti érték
init : Statusz='Kezdo'
--Származtatott érték
derive : if Pont>10 and Pont<=20 then
Statusz='Halado'
Endif
```

Definíciók

Az OCL nyelv két lehetőséget nyújt kifejezések másképp történő elnevezéséhez, amivel később hivatkozni lehet rá. Az egyik a let-in kifejezéspáros amiben csak az in kulcsszó után lévő kifejezésben lehet a definiált elnevezéssel hivatkozni, a következő OCL típusban már nem lesz érvényes a definíció.

```
context Szemely inv:
let bevetel:Integer=self.munka.fizetes->sum()
in
if munkanelkuli then
bevetel<100
else
bevetel>=100
endif
```

A másik lehetőség a definiálásra a def kulcsszóval történik. Ha ezzel van definiálva akkor már az egész classifier-ben használható.

```
context Konyv
def: oldalszam:integer = self.oldal->sum()
def: cim='Hello world'
```

Látható, hogy az OCL többféle megszorítás kifejezésére és leírására is lehetőséget ad. Kifejezőkészlete azonban nem hasonlítható össze a programozási nyelvek kifejezőkészletével. Nem is célja egy ilyen fokú komplexitás biztosítása. Fejlesztése folyamatos.

13. Case eszközök

Az UML manapság egy elterjed modellező nyelvnek számít. Szabványos volta, a benne rejlő következetesség, a modellezési lehetőségek sokoldalúsága kedveltté teszi. Ez hatással van mind a nyílt forráskódú, mind a kereskedelemben megvásárolható modellező rendszerek terjedésére. Egyre több olyan szoftver kerül piacra, vagy a világhálóra, amelynek segítségével könnyen megismerhető, kipróbálható az UML. Ezek az úgynevezett CASE (Computer-aided software engineering) eszközök[9], amelyeket kifejezetten a szoftverfejlesztés támogatására fejlesztenek. A legtöbb rendszer támogatja a modellvezérelt alkalmazásfejlesztést is, tehát a modell egy fordító segítségével kóddá alakítható. Az OMG is támogatja ezeket a fejlesztéseket, több világszerte ismert céggel együtt dolgozik a szabvány továbbfejlesztésén. Ez mind a cégek, mind az OMG számára előny, hiszen a cégek első kézből értesülhetnek az újdonságokról, másrészt az OMG gyakorlati tapasztalatokat szerez az UML használhatóságáról, hiányosságairól. Az OMG által is javasolt UML támogató eszközök listája megtalálható a http://www.uml.org oldalon. További információkat, bemutatókat, profilokat, érdekességeket találhatunk ugyanezen és a kapcsolódó weboldalakon. Javaslom tanulmányozásra ezeket az anyagokat, nagyban segíti az UML mélységeinek megismerését.

[9] http://en.wikipedia.org/wiki/Computer-aided_software_engineering

14. Gyakorló feladat

A tananyag nem tartalmazza az esettanulmány teljes modelljét. Segíti a modellezésbe való gyakorlat megszerzését, ha az olvasó a hiányzó diagramokat és táblázatokat elkészíti, követve a lehetséges tervezési lépéseket. A rendszer új nézőpontokból történő megismerése támogatja a rendszertervezésben használt modellek részletes megismerését.

15. Ajánlott irodalom

[1] R. Mile, K. Hamilton, Learning UML 2.0, O'Reilly, 2006.
[2] H. Störrle, UML 2 – Unified Modeling Language, Panem Könyvkiadó, 2007.
[3] Unified Modeling Language™ (UML®) - Object Management Group specifications: http://www.omg.org/spec/UML/
[4] Object Constraint Language, OMG Available Specifications: http://www.omg.org/spec/OCL/
[5] Object Management Group: http://www.omg.org/

Ábrák, táblázatok jegyzéke

Ábrák

Táblázatok