DEBRECENI EGYETEM
INFORMATIKAI KAR

Az UML gyakorlati alkalmazásának bemutatása az AutoWorld rendszer tervezésén keresztül

Témavezető:
Pánovics János
egyetemi tanársegéd

Készítette:
Hegedűs József
programtervező informatikus

Debrecen, 2010

Tartalomjegyzék

1. BEVEZETÉS 4

1.1 Modellezés 4

2. MI AZ UML? 6

2.1Az UML és felépítése 6

2.2Az UML-hez vezető út 7

2.3Az UML létrejötte és története 8

2.4A bemutatott UML eszközök 10

2.4.1 Diagramok 10

2.4.2 Kiterjesztési mechanizmusok 11

3. A TERVEZENDŐ RENDSZER 13

4. STRUKTURÁLIS DIAGRAMOK 15

4.1 Osztálydiagram 17

4.1.1 Elemzési osztálydiagram 17

4.2.1 Tervezési osztálydiagram 22

4.3.1 Megvalósítási osztálydiagram 30

4.2 Objektumdiagram..... 32

4.3 Csomagdiagram 35

4.4 Komponensdiagram 39

4.5 Kihelyezési diagram..... 41

5. VISELKEDÉSI DIAGRAMOK 43

5.1 Használati eset diagram..... 43

5.2 Állapot-átmenet diagram..... 47

5.3 Tevékenységdiagram..... 49

5.4 Interakció diagramok 52

6. ÖSSZEFOGLALÁS 56

7. IRODALOMJEGYZÉK 57

1. BEVEZETÉS

1.1 Modellezés

A szoftverfejlesztés vonatkozásában egy modell nem más, mint egy leképezés. Ez a leképezés lehet egy elképzelt rendszer különböző folyamatainak és elemeinek egy előzetes, vázlatos megadása, leírása. A modellt ekkor nevezhetjük korai elemzésnek vagy tervezésnek is. Segítségével könnyebben átláthatjuk saját elképzelésünket, könnyebben felfedezhetjük az esetleges hibáinkat, valamint mások is egy átfogó és áttekinthető képet kapnak magáról a fejlesztendő szoftverről, legyenek ők akár a fejlesztő csapat tagjai, megrendelők vagy a leendő felhasználók.

A modell egy igen fontos és nagyon hasznos tulajdonsága tehát az áttekinthetőség. Az áttekinthetőséget az garantálja, hogy a modellezés eredménye az a vázlatos terv, amely nem más, mint az alkalmazás absztrakt modellje. Ugyanis a modellek a programrészletekhez viszonyítva egy magasabb absztrakciós szinten ábrázolják rendszerünket, amely ábrázolásban nem jelennek meg a konkrét megvalósítási technikák zavaró részletei, amelyet lehetséges, hogy a megrendelő és a leendő felhasználó nem is értene. Ezáltal mentesülnek a tényleges, bizonyos nyelven megírt kódok megértése alól, mégis egyértelműen látják, mi is valósult meg az elképzeléseikből.

A modell másik fontos tulajdonsága az, hogy átfogó képet ad az elkészülendő rendszerről azáltal, hogy már a kódolás előtt a rendelkezésünkre áll a teljes rendszer. Természetesen nem futtatható állapotban, de így is még a tényleges programozás előtt ellenőrizhető, hogy a terv megfelel-e az előzetesen megadott követelményeknek. Az esetlegesen itt felmerülő új vagy eddig figyelmen kívül hagyott követelmények is könnyen kezelhetőek a modellünk újratervezésével. A modell alapján felfedezhetünk egyéb hiányosságokat és hibákat is, amelyeket szintén viszonylag egyszerűbben tudunk korrigálni. Könnyen belátható, hogy az itt végbemenő változtatások sokkal kevesebb költségbe és energiába kerülnek, mint ha a már kész programunkat kellene módosítani.

A modell alapján megalapozott, dokumentált döntést tudunk hozni a konkrét megvalósításról. A leképezés alapja azonban lehet egy már meglévő alkalmazásrendszer is, ekkor egyfajta dokumentálásról beszélhetünk. Ekkor ugyanúgy elkészül a rendszer modellje, azonban itt már csak a tényleges rendszerre kell figyelni, nem kell hiányosságokat és hibákat keresni, pusztán a rendszer egy vázlatosabb másolatát kell elkészíteni. Azonban az elkészült modell segítségével ekkor már eldönthető, hogy mennyire is hasznos számunkra az éppen használatban lévő alkalmazás. Az illetékes vezetők így könnyebben dönthetnek arról, hogy kell-e fejleszteni a rendszert vagy sem, esetleg szükséges-e lecserélni azt. Az elkészült modell óriási segítséget nyújt az újratervezés esetén, hiszen felhasználva és tanulva hibáiból könnyebben és gyorsabban elkészíthető az új rendszer egy modellje.

Modelljeink áttekinthetőbbé és érthetőbbé válnak, ha ábrázolásukhoz grafikus jelölésrendszert, azaz diagramokat használunk. Egy diagram bizonyos, előre definiált, a diagramon értelmezhető elemeket és a közöttük lévő viszonyokat tartalmazza. Az ilyen diagramok egy gráfot alkotnak, ahol a csomópontok megfelelői a diagramon szereplő, különböző elemek; míg a gráf élei az elemek közötti kapcsolatok megfelelői.

Az elmúlt néhány évtizedben rengeteg grafikus jelölésrendszer kialakult. Azonban az egyik ilyen modellező nyelv jelentősége akkora volt, hogy megjelenése után nem sokkal szabvány lett belőle. Ez az UML.

2. Mi az UML?

2.1 Az UML és felépítése

Az UML az Unified Modelling Language rövidítése, amelynek magyar fordítása: Egységesített modellező nyelv. Az UML az objektumorientált alkalmazásfejlesztés grafikus tervező, elemző és dokumentáló eszköze. Egyaránt szolgálja a szoftverfejlesztők egymás közötti, és a felhasználók felé irányuló kommunikációt is. Egyik legnagyobb előnye az, hogy egységesített. Egységes volta abból adódik, hogy története során a vezető módszertanok „összegyúrásából" jött létre. Az UML-t a kanonizálás folyamata jellemezte és jellemzi mai is, hiszen a jól bevált koncepciók gondosan válogatott, egymás mellett tökéletesen működő, egymást kiegészítő részeit építi be fogalom és eszközrendszerébe. Mára a világ legtöbb vezető szoftvervállalata is felismerte az UML jelentőségét, foglalkozik a szabvány továbbfejlesztésével, így az utóbbi években az UML eszközkínálata nagymértékben megnőtt. Azonban ez magával vonta azt is, hogy az UML-nek különböző résznyelvei jöttek létre. Ezek a résznyelvek azonban integráltak, közös infrastruktúrájuk és a metamodellel közös koncepcionális vonatkoztatási keretük van. Egy nagyon fontos tulajdonsága a jelölésrendszernek, hogy szabványosított. Ez lehetővé teszi a modellek és modellezési ismereteink hordozhatóságát, valamint segít az UML terjeszkedésében. Az utóbbi években bár még mindig érték kritikák, a többség nem vonja kétségbe szükségességét.

Az UML mint a neve is elárulja modellező nyelv. A modellező jelző arra utal, hogy a teljes implementáció nem ábrázolható vele, csupán az alkalmazás vázlata. Azonban ez a vázlat is tökéletesen elegendő számunkra akkor, ha nagy és bonyolult rendszereket tervezünk vagy fejlesztünk. Ilyenkor óriási segítséget nyújt, ugyanis átláthatóbbá, kezelhetőbbé, érthetőbbé, és ezáltal gyorsabbá teszi munkánkat. A gyorsabb fejlesztés vagy tervezés nem abból adódik, hogy módszert ad az egyes lépésekre, mivel az UML nem módszertan, csupán a modellek ábrázolására alkalmas jelölésrendszer. Alapvetően grafikusnyelv, de a vizuális eszközökön kívül tartalmazhat szöveges részeket is.

Az UML felépítése lényegében egy metamodellezési megközelítésből indul ki, amely azt jelenti, hogy minden konkrét rendszer egy modell példánya, és minden modell egy metamodell példánya, míg a metamodell egy meta-metamodell példánya, amely meta-metamodell a MOF (Meta Object Facility). A hierarchia úgy zárul le, hogy a MOF saját maga példányként van definiálva.

absztrakciós réteg	eszköz	példa programozási nyelvre	példa modellező nyelvre
M3	meta- metamodell	EBNF	MOF
M2	metamodell	a Java-nyelvtan	UML
M1	modell	egy Java-program	egy konkrét rendszer
M0	rendszer	egy Java-program végrehajtása	a futási idő állapota egy konkrét rendszer működése közben

1. ábra Az UML szintszerkezete

2.2 Az UML-hez vezető út

A ’70-es években felismerték, hogy nagyméretű és bonyolult rendszereket nem lehet egy olyan eljárás nélkül létrehozni, amely meghatározná a fejlesztés lépeseit, előírná azok sorrendjét. Ezáltal megadva egy olyan technológiai sort, amely biztosítaná megfelelő hozzáértéssel a sikert, valamint az emberi tényező bizonytalanságát csökkentené és így a fejlesztés követhetővé és ellenőrizhetővé válna. Az objektumorientáltság megjelenésével egyre többen vélekedtek úgy, hogy ez az új szemlélet alkalmazható rendszerfejlesztési módszerként is. Először az objektumorientált tervezés (OO design) jelent meg, majd az elemzés (OO analysis). Ezt alapul véve az évek folyamán sok módszer és jelölés alakult ki.

Azonban néhány kiemelkedett közülük, ezek később fontos szerepet is játszottak a későbbi szabvány létrejöttében. Grady Booch alkotta meg elsőként módszerét, amelyben az ADA nyelv bemutatása volt a cél. Booch felhasználta Abbot ötletét: a feladat szöveges megfogalmazásakor legyen egy egyszerű nyelvtani elemzés. Booch mindemellett programjai alapszerkezetét diagramokkal ábrázolta, ezzel is egyszerűbbé téve a megoldás megértését. A hangsúlyt a tervezési fázisra fektette. 1991-ben megjelenik az OMT, amely szintén módszertan és jelölés egyben. Az OMT megalkotásának vezéralakja James Rumbaugh volt. Az OMT leírásakor mindent közérthetően és egyszerűen definiáltak, amely segített népszerűsítésében. Rumbaugh munkássága folyamán egy konzisztens jelölésrendszer jött létre, amely szintén használta az Abbot-féle szövegelemzést, és az analízis fázisához alkotott kiemelkedőt. Egy szintén meghatározó módszer és jelölésrendszer jött létre Ivar Jacobson vezényletével. Jacobson és társai az Objectory, azon belül az OOSE megalkotásánál elsősorban a gyakorlati megközelítésnek tulajdonítottak nagyobb jelentőséget. Ezt alátámasztja újításuk is, hiszen hasonlóan a többi OO módszerben is megjelenő elemek mellett módszerükben helyet kap a használati esetek (use case) technikája is. Lényege az volt, hogy megadjuk a felhasználó és a rendszer közötti interakciókat, összefoglalva ezáltal a rendszerrel kapcsolatos követelményeket. Az évek folyamán azonban egyre inkább felismerték, hogy egységesítésre lenne szükség.

2.3 Az UML létrejötte és története

Az egységesítés szükségessége a 1990-es évekre egyre nyilvánvalóbbá vált. Ekkor már több, a szoftver teljes életciklusát átfogó, hatékony módszertan létezett. A módszertani változatok több szempontból is hasonlóak voltak, viszont akadt jócskán különbség is köztük. Egyik legnagyobb különbség volt, hogy eltérő jelölésrendszert alkalmaztak. A fejlesztők dolgát ez igencsak megnehezítette, hiszen más-más, talán egymást követő projektben, amelyben munkálkodtak, esetleg más-más jelölésrendszert kellett használniuk. Több kísérlet is indult az egységesítésre, azonban mind Booch, mind az OMG törekvései ez ügyben eleinte sikertelenek voltak. Minden módszertan ősatyái ragaszkodtak a saját elképzelésükhöz, ezért ezt az időszakot a „módszerek háborújának” is nevezték.

1994-ben az OOPSLA-n[1] Booch és Rumbaugh bejelentette, hogy vége a „háborúnak”. A bejelentés lényege az volt, hogy Rumbaugh csatlakozik Booch-hoz és cégéhez a Rational-hez, és közösen dolgoznak ki egy új módszert. Az egyesülés eredményét a ’95-ös OOPSLA-n mutatták be, amely az UM (Unified Method), az Egységesített módszer volt. Érdekessége többek között az volt, hogy a 0.8-as verziószámot kapta; ezzel is arra utalva, hogy majd az 1.0-ás verziót szeretnék szabványosítani. Ezen a konferencián jelentették be, hogy a Booch Rumbaugh duóhoz csatlakozik Jacobson is. Jacobson újabb ötleteket és lendületet hozott a projektbe. Munkásságuk során szerzett tapasztalataiknak köszönhetően, valamint más módszertanok előnyeit felhasználva ’96-ban kiadták a 0.9-es változatot, amelyet azonban már UML-nek neveztek el. Az UML sok kritikát kapott, ugyanis módszertani elemeket nem tartalmazott, ezt kivédvén a Jacobson-féle Objectory-t ajánlottak új jelölésrendszerünk mellé, mint módszertant. Mivel tudták, közel vannak a szabványosítható verzióhoz, legújabb verziójukat neves informatikai cégeknek ajánlották fel tesztelésre. Ez a kísérlet igencsak sikeresnek bizonyult, hiszen a cégektől kapott tapasztalatokat felhasználva 1997 januárjára elkészült az 1.0-ás verzió, amelyet már késznek titulált a három vezető egyéniség. Azonban kisebb hibák még ekkor is voltak, például a metamodell oldaláról. Ezeket kijavítva még ’97 szeptemberére elkészült az 1.1-es verzió, amelyből már tényleges szabvány lett. Egy évre rá elkészült az UML-hez javasolt módszertan a RUP, azonban később több más szoftvercég is készített saját módszertant az UML-hez.

Az azóta eltelt időben gyors egymásutánban jelentek meg az egyre bővebb és korszerűbb 1.x verziók. 2001-ben megjelent az UML 1.4, majd 2002-ben az UML 1.5. Az 1.5-ös verzió olyan eszközöket tartalmazott, amelyek segítségével az UML által formalizált modellből kódot lehetett generáltatni. Az utolsó változat az UML 2.0 volt, amely 2005-ös megjelenésekor OMD szabvány lett. Dolgozatomban a jelenleg is érvényben lévő verzió leggyakrabban használt eszközeit szeretném bemutatni.

[1] Object-Oriented Programming, Systems, Languages and Applications; Az Egyesült Államok legnagyobb, az objektumorientáltsággal foglalkozó konferenciája.

2. 4 A bemutatott UML eszközök

2.4.1 Diagramok

Az UML-ben diagramok segítségével jeleníthetőek meg a szoftverfejlesztés során szükséges információk és összefüggések. A diagram olyan gráfnak tekinthető, melynek csomópontjai az UML eszközei közé tartozó elemek, míg élei az ezen elemek között fennálló kapcsolatokat. Számos diagramtípus létezik, melyek gyakran kiegészítik egymás mondanivalóját, néha viszont ugyanazt a területet mutatják be más szemszögből. Ezáltal egy diagram eleme megjelenhet egy másik diagramon is, bár ott lehet más funkcióban. Az UML a diagramokat két nagy csoportba sorolja: strukturális és viselkedési diagramok. A strukturális diagramok a rendszer felépítését, belső szerkezetét mutatják be; míg a viselkedési diagramok inkább a működését és a folyamatait ábrázolják. A következő csoportosítás a dolgozatban felhasznált diagramokat helyezi el az előbb említett két kategória valamelyikében.

Strukturális diagramok:

- Osztálydiagram
- Objektumdiagram
- Csomagdiagram
- Komponensdiagram
- Kihelyezési diagram

Viselkedési diagram:

- Használati eset diagram
- Állapot-átmenet diagram
- Tevékenység diagram
- Interakció diagramok

2.4.2 Kiterjesztési mechanizmusok

Az UML rengeteg eszközt tartalmaz, melyek segítségével a rendszer egyes részleteit be lehet mutatni. Azonban hiába az óriási eszköztár, vannak olyan speciális esetek, amikor az UML vagy nem kínál hatékony megoldást, vagy egyáltalán nem tartalmaz megoldást az adott helyzetre. Az UML lehetőséget ad a speciális esetek egy olyan kezelésére, hogy a szabványos elemei a szabványban nem szereplő elemekkel egészíthetőek ki. Ennek segítségével a szabványos jelölésrendszer keretein belül pontosabb modell adható meg az aktuális rendszerekhez.

Az UML kiterjesztési mechanizmusai:

- sztereotípia
- megszorítás
- hozzárendelt értékek

Sztereotípia

A sztereotípia olyan kiterjesztési mechanizmus, melynek segítségével a meglévő UML-beli elemek mellé új elemek hozhatóak létre, sokszor pont egy UML-beli elem átdefiniálásával. Ezen kívül sztereotípia felhasználásával a meglévő elemek kategorizálását is elvégezhetjük. Egy elemhez úgy rendelhetünk sztereotípiát, hogy a neve felett elhelyezzük a sztereotípia szövegét. A sztereotípia úgy jelölhető, hogy dupla francia idézőjelek közé helyezzük annak karaktersorozatát. Azonban a sztereotípia nemcsak szöveges formában jelenhet meg, hanem szimbólummal vagy ikonnal jelölve is. Ha ikonokat használunk az ábrázoláshoz, megkönnyítjük a későbbi olvasó számára a diagram megértését.

Megszorítások

Megszorítások használatával az elkészítendő modellt tovább lehet pontosítani azáltal, hogy az elemekre vonatkozóan megadható korlátozás, pontosítás vagy bizonyos mértékű

változtatás. Megszorítás lehet egyetlen tulajdonság, de leírhat több elem közötti kapcsolatot is. Az adott UML-beli elem elnevezése alatt vagy mellett adható meg a megszorítás, amely többnyire szöveges formában van rögzítve. Ekkor a megszorítás szövege kapcsos zárójelek között van elhelyezve. Megszorítás azonban megadható valamilyen formalizmus vagy formális nyelv segítségével is, de megadható egyszerűen egy megjegyzés tartalmaként is. A formális megadásra ajánlott az OCL nyelv.

Hozzáadott érték

Meglévő UML elemek nevéhez értékeket rendelhetünk. A fejlesztéssel kapcsolatos információkat például hozzáadott értékek segítségével adjuk meg. Ilyen lehet például a szerző vagy a verziószám is.

A tervezendő rendszer

Az **AutoWorld** egy számítógépes rendszer, amely egy autószalon működését segíti. Az autószalon autók eladásával és kölcsönzésével foglalkozik. A rendszer mindkét területet képes kezelni. Nyilvántartja az éppen készleten lévő autókat, azok fontos tulajdonságaival és paramétereivel együtt. Az autókat a rendszer képes megkülönböztetni aszerint, hogy azok eladóak-e vagy kölcsönözhetőek-e.

A rendszert három személy használhatja:

- ügyfél
- eladó
- üzletvezető

Az ügyfél a rendszer webes felületén keresztül tudja megtekinteni az éppen rendelkezésre álló autókat, külön listázva a kölcsönözhető és az eladó autókat. Az ügyfeleknek lehetőségük van az autókat lefoglalni, ha meg szeretné vásárolni. Ha kölcsönözni szeretne, lefoglalhatja a kölcsönözendő gépkocsit, hogy mások ne béreljék ki hamarabb. Az előjegyzés során megadhatja, hogy melyik autót, mikortól és mennyi időre szeretné bérelni. Ha esetleg később meggondolná magát az ügyfél, lehetősége van az előjegyzés visszavonására.

Az eladó a rendszer segítségével megtekintheti az autók listáját. Az eladó feladata és jogköre ezen felül igen széleskörű. Az ő feladata, hogy továbbítsa azon autók listáját az üzletvezető felé, amelyeket hiány miatt meg kell rendelni. Ezen kívül rögzíthet kölcsönzéseket és eladásokat, valamint törölheti azokat. Feladata még a vásárlók foglalásainak véglegesítése, azaz a kért eladás rögzítése. Az ügyfelek kölcsönzési igényét is elbírálja, és ha lehetséges a kölcsönzés, akkor véglegesíti a kölcsönzést. A kölcsönzés lejárta után az eladó feladata, hogy a rendszeren belül visszavegye a gépkocsit.

Az üzletvezető felelős az autók megrendeléséért, az újonnan érkezett autók felviteléért a rendszerbe, amelyek majd kölcsönözhető vagy eladó autók lesznek. A nyilvántartott autókat törölheti, az esetleges leselejtezés esetén. A felvitelen túl képes az autók árainak módosítására is. Árleszállítás esetén, akár az összes gépkocsi árát egyszerre tudja módosítani, ha megadja

az árengedményt százalékos formában. Az AutoWorld segítséget nyújt abban is, hogy különböző kimutatásokat készítsen.

A rendszer nyilvántartja még az autókon kívül a megrendeléseket, foglalásokat, eladásokat és a kölcsönzéseket is. A rendszerbe minden felhasználónak regisztrálni kell.

A kereskedés a régi, visszatérő ügyfeleit megbecsüli, ezért személyre szabott kedvezményt is nyilvántart, amely bármely kölcsönzés vagy vásárlás esetén megilleti az ügyfelet. A kedvezmény a kölcsönzési összeg, vagy a vásárlási ár bizonyos százalékának elengedését jelenti.

4. STRUKTURÁLIS DIAGRAMOK

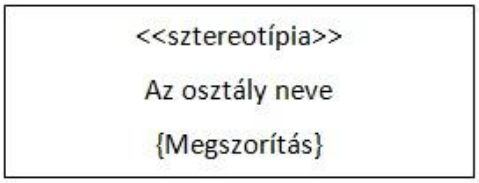
4.1 OSZTÁLYDIAGRAM

Egy alkalmazás alapszerkezete úgy adható meg, ha megkeressük az alkalmazást felépítő alapelemeket. Ezután megadjuk azok tulajdonságait, valamint azt, hogy milyen viszonyban állnak egymással. Az objektumorientált szemlélet szerint rendszerünk alapjai az objektumok. Azonban ezen objektumokat csoportosíthatjuk, osztályozhatjuk, ezáltal létrejön az objektumokhoz tartozó osztály, ami a rendszer egy fontos alapeleme, fogalma lesz. Az osztálydiagram ezeket a fogalmakat összegyűjti, és a köztük fellépő kapcsolatokkal együtt grafikusan ábrázolja. Ez a diagram már a belső tervezést szolgálja. A felhasználók és megrendelők ezen a szinten már nem vesznek részt a fejlesztésben, hiszen valószínűleg nem rendelkeznek kellő ismerettel ezen a területen. Az osztálydiagramok jelen vannak csaknem a teljes szoftveréletciklusban. Fontosabb szerepük azonban az elemzés, a tervezés és a megvalósítás fázisában van.

4.1.1 Elemzési osztálydiagram

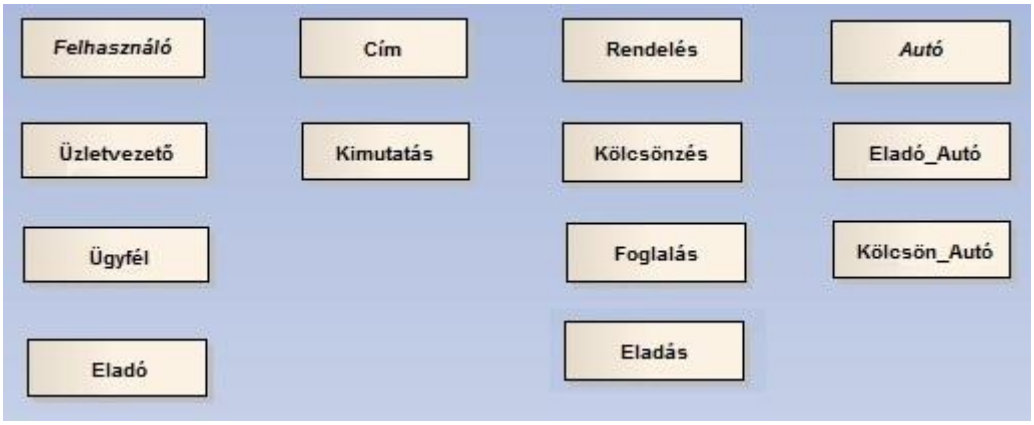
Elemzés során az osztályok annak a szakterületnek a fogalmait definiálják, amelyben az alkalmazás működni fog. A megfelelendő osztályok létrehozásához elengedhetetlen a szakterületen való kellő mértékű tájékozottság. Ennek hiányában a szükséges ismereteket a területen dolgozó szakértőktől szerezhetjük meg, például interjúk formájában. Elemzési szinten elfogadott, hogy minden lényeges szakmai fogalomnak egy osztály felel meg.

Az UML-ben az osztályt egy téglalap jelöli. A téglalap felső részébe középre igazítva írjuk az osztály nevét. Ezt a részt névrésznek is nevezzük, amely más modellelemeknél is megjelenhet. Az osztályok névrészében, a név alatt megadhatunk az osztályhoz tartozó megszorítást, valamint az osztály neve felett sztereotípiát is. Erre mutat egy példát a 2. ábra.



2. ábra Egy osztály általános megadása

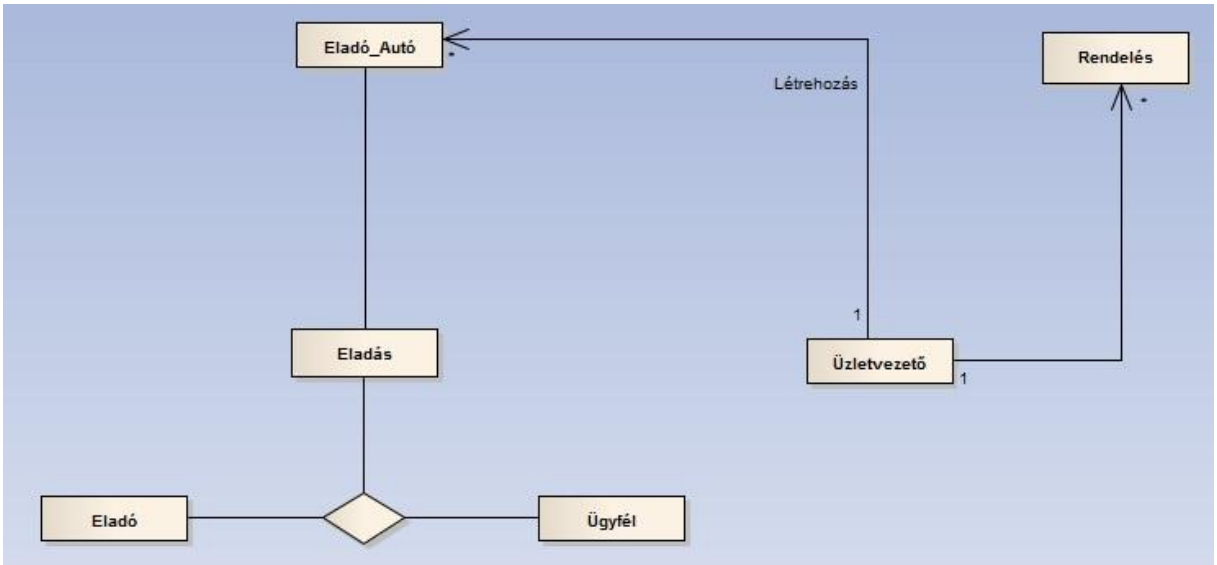
Mint azt már leírtam az osztályok a szakterülethez tartozó fogalmak megfelelői. Az AutoWorld alkalmazás esetében fogalmakként megjelennek egy autókereskedés és kölcsönző esetében felmerülő fogalmak. A kereskedés mint üzlet, természetesen rendelkezik üzletvezetővel és eladókkal, valamint azon ügyfelekkel akik ott vásárolni vagy kölcsönözni szeretnének. Az említett szereplők lesznek az alkalmazás felhasználói. Külön fogalomként jelenik meg a felhasználók címe. Az autószalon autóival kereskedik. Az autókat azonban csoportosíthatjuk aszerint, hogy melyeket lehet kölcsönözni, és melyek az eladóak. Az üzletvezető rendelhet új autókat a szalonba, valamint kimutatást készíthet, amely szintén megjelenik, mint osztály. Az autók lefoglalhatóak, kölcsönözhetőek és eladhatóak. Ezek alapján a 3. ábra tartalmazza az AutoWorld összes osztályát, ezáltal lényeges fogalmát.



3. ábra Az AutoWorld osztályai

Ezen fogalmak között – és így osztályok között is – léteznek különböző kapcsolatok. Két osztály között fennálló legalapvetőbb kapcsolat az **asszociáció**. Ez a típusú kapcsolat nem más, mint az osztályok objektumai közötti lehetséges strukturális kapcsolat. Az UML-ben bináris, ternáris és magasabb rendű asszociáció létezik. Bináris asszociáció két osztály közötti kapcsolat, amelyet az UML a két osztályt összekötő vonallal jelöli. A 4. ábrán látható, hogy az Eladás és az Eladó_Autó között bináris asszociációs kapcsolat áll fenn. A **ternáris asszociáció** három osztály között, míg a magasabb rendű háromnál több osztály között áll

fenn. Mindkét asszociációt az UML egy rombusszal jelöli, amelyet vonalak kötnek össze a kapcsolt osztályokkal. Az Eladás, az Eladó és az Ügyfél között ternáris asszociációs kapcsolat áll fenn, ezzel is hangsúlyozva, hogy az asszociációban résztvevő mindhárom egyenrangú szereplő.



4. ábra Elemzési osztálydiagram részlete

Léteznek olyan osztályok is, melyek identitással rendelkező asszociációk; ezeket asszociációs osztályoknak nevezzük. Az asszociációs osztályokhoz saját attribútum és metódus is rendelhető. Az asszociáció névvel is ellátható, amely konkretizálja a kapcsolat lényegét. A 4. ábra szerint az Üzletvezető hozhatja létre az Eladó_Autó egy példányát. A művelet csak egy irányba mehet végbe (hiszen az Eladó_Autó nem hozhat létre új Üzletvezetőt), így ezt az asszociáció megfelelő végén elhelyezett nyíllal jelölhetjük. Az asszociáció végei egy bizonyos szerepet töltenek be a kapcsolatban, ezért ezeket szerepköröknek nevezhetjük. A szerepköröknek nevet is adhatunk. A szerepköröket multiplicitással (számossággal) is elláthatjuk, amely jelzi, hogy egy konkrét kapcsolatban egy osztály hány objektuma vesz részt. A 4. ábrán az Üzletvezető és a Rendelés kapcsolata multiplicitással rendelkezik, amely azt hivatott jelezni, hogy egy Üzletvezetőhöz több Rendelés is tartozhat, ami megfelel a valóságnak. Hasonlóan egy Üzletvezető létrehozhat több Eladó_Autó-t is.

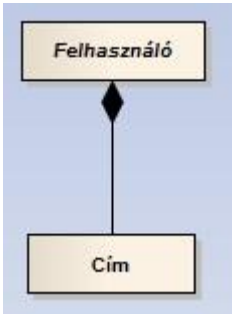
A számosság továbbá megadható a következő három mód egyike szerint:

- egy számérték (természetes szám)

- a..b formában megadott értéktartomány
- az előző kettő vesszővel elválasztott sorozata

Intervallumok határán tetszőleges természetes szám állhat, beleértve a 0-át is. Megkülönböztetett jelentésű a * karakter, amely a végtelent jelöli; intervallum esetén pedig, amikor 0..* szerepel, tetszőleges számú multiplicitást jelent.

Az asszociációnak létezik két speciális esete: kompozíció és aggregáció. Az aggregáció a két modellelem között fennálló rész-egész viszonyt jelöli. Az „egész" logikailag tartalmazza a „részt". Az aggregációt az asszociáció „egész" szerepkörénél elhelyezett rombusszal jelöljük. Aggregáció esetén azonban az UML nem követeli meg, hogy az „egész" törlése esetén törlődnie kell a „résznek" is. Ez némi bizonytalanságot jelent, ezért az UML később bevezette a kompozíció fogalmát, amely az aggregáció egy erősebb változata. Jelölése az „egésznél" elhelyezett fekete rombusz. Kompozíció során a „rész " egyértelműen az „egész" alkotóeleme, így a „rész" nem létezhet az „egész" nélkül. Tehát ha egy „egész" törlődik, törlődik a hozzá tartozó „rész" is. „Rész" akkor törlődhet önállóan, ha a számosság ezt megengedi. Kompozíció csak bináris kapcsolat lehet. Rendszerünk esetén kompozíciós kapcsolat áll fenn a Felhasználó és a Cím között, ahol a Cím „része" a Felhasználónak. A Felhasználó létezése nélkül ugyanis a Cím nem létezhet. Ugyanakkor, ha törlődik egy Felhasználó, a hozzá tartozó Cím is törlődni fog. A kapcsolat jelölését az 5. ábra mutatja be.

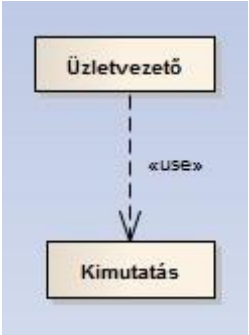


5. ábra Kompozíció

Osztályaink között fennállhat **függőség**. Egy osztály függ egy másik osztálytól, amikor az osztály változását vonhatja maga után a másik osztály megváltozása. Tipikus eset, amikor az egyik osztály hivatkozik a másik osztály egy mezőjére. A függőséget egy szaggatott nyíl jelzi, amely a függő felé mutat. A függőség jellegét sztereotípiaként adhatjuk meg. Lássuk a leggyakoribbakat:

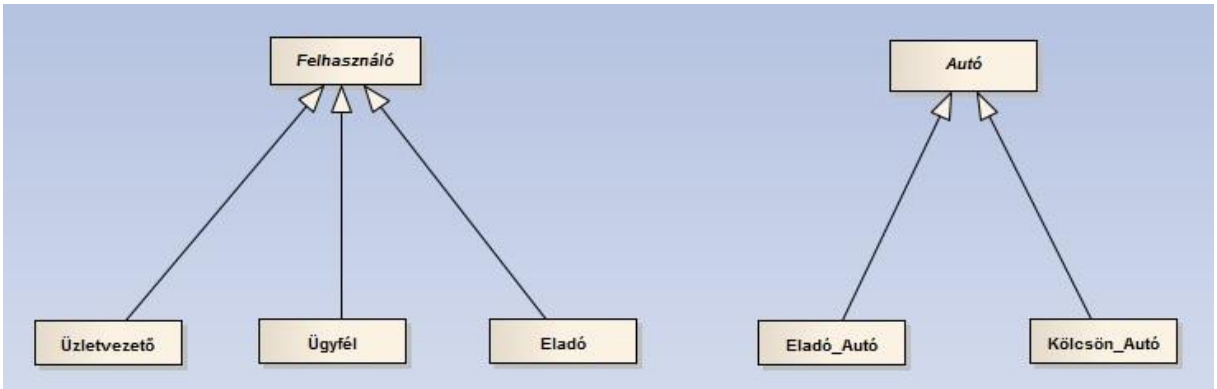
- <<use>>: a függő osztály definíciója során felhasználja a másik definícióját
- <<friend>>: a függő osztály eléri a másik osztály védett vagy saját jellegzetességeit
- <<call>>: a függő osztály meghívhatja a másik metódusát
- <<instanceOf>>: a függő objektum az osztály konkrét példánya

Az AutoWorld rendszer osztályait tekintve észrevehető, hogy függőségi viszony áll fenn az Üzletvezető és a Kimutatás között. Ezt a 6. ábra mutatja be. A függőség abból adódik, hogy a Kimutatás definíciója során kötődik az Üzletvezetőhöz. Ez egy tipikusan <<use>> sztereotípiával megadott függőség.



6. ábra Függőség

Az objektumorientáltság egyik központi fogalma az **öröklődés**, amely általánosításként megjelenik az UML-ben is. Öröklődés során beszélhetünk ősosztályról és a hozzá tartozó alosztályról. Az UML az öröklődést egy fehér hegyű nyíllal jelzi, amely az alosztály felől az ősosztály felé mutat. Az tervezendő rendszer esetében több öröklődési viszony is fellelhető.



7. ábra Öröklődési viszonyok

A 7. ábra jól szemlélteti, hogy a rendszert később használó Ügyfél, Eladó és Üzletvezető is felhasználója lesz az alkalmazásnak, és ezt a tulajdonságot kiemelve egy Felhasználó osztály jön létre. A másik két öröklődési viszony abból adódik, hogy ugyan autókkal foglalkozik a

szalon, azonban az egyes tevékenységekhez tartozó autók (mint például kölcsönzés vagy eladás), speciális autók. Így a Kölcsön_Autó és az Eladó_Autó az Autó pontosításaként jönnek létre.

Ha egy osztályhoz több alosztály is tartozik, akkor ezeket összevonhatjuk egy általánosításhalmazba. Ezután az ebben szereplő alosztályokra egyszerre tehető megszorítás. Ezt az UML az általánosítási kapcsolatok összevonásával, egy közös fehér hegyű nyíllal ábrázolja. Az általánosításhalmazoknak két tulajdonsága jelölhető a közös nyíllal:

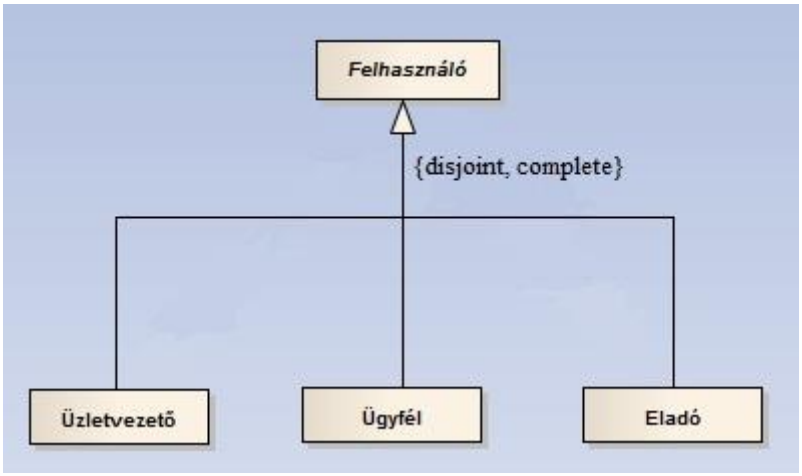
- disjoint – overlapping:

  Disjoint esetén az általánosításhalmazok belül az általánosítások diszjunktak, míg az overlapping ennek ellentéte.

- complete – incomplete:

  Ha complete szerepel az ábrán, jelzi, hogy az ősosztály minden példánya ebben az általánosításhalmazban lévő egyik alosztály példánya. Ennek ellentéte az incomplete.

A 7. ábrán látható öröklődések esetén a Felhasználók alosztályai is összevonhatóak. Ugyanez megtehető lenne az Autó alosztályai esetén is. A 8. ábra csak az elsőt mutatja be.

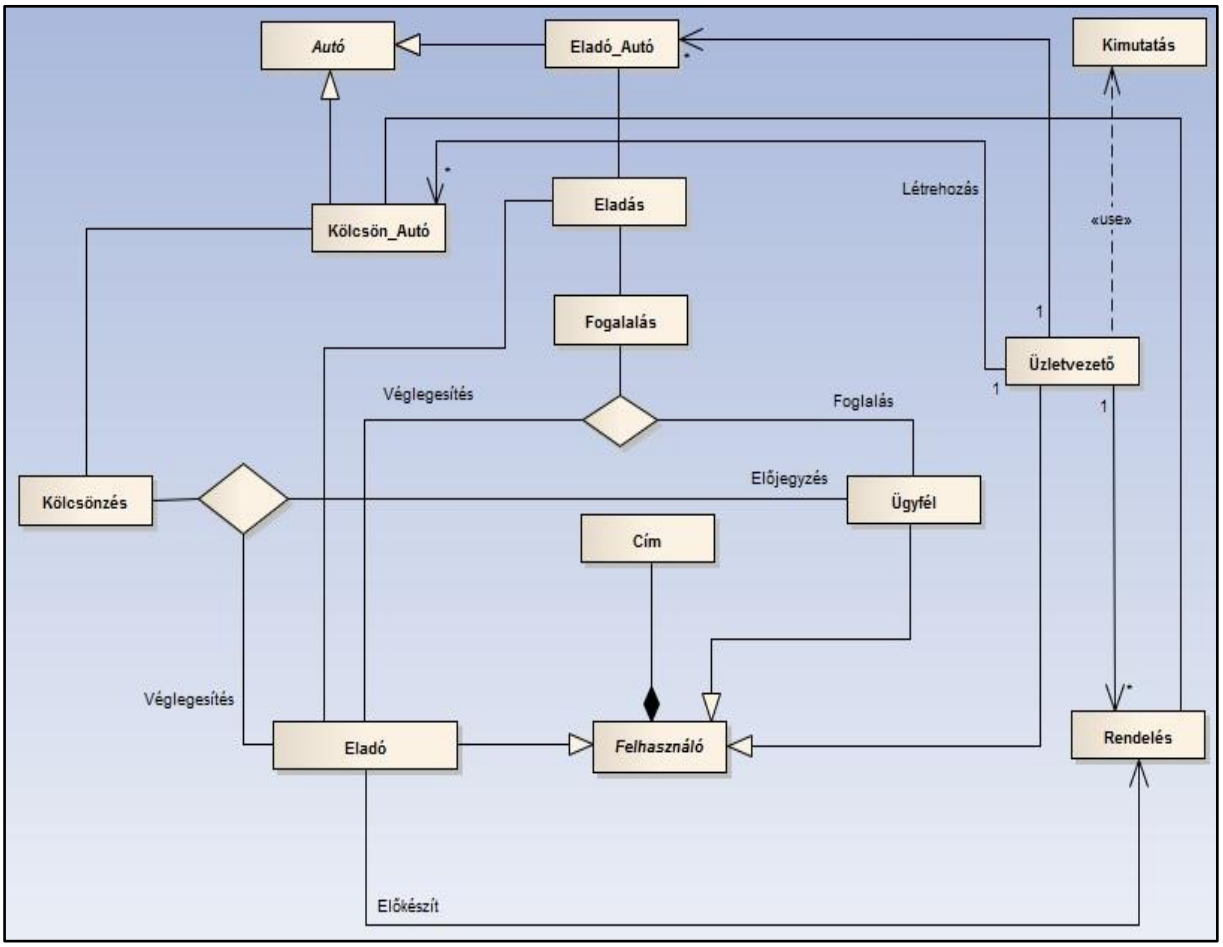


8. ábra

Az UML mint modellező nyelv többszörös öröklődést vall, mely lehetővé teszi, hogy egy alosztálynak több ősosztálya legyen. Azonban gyakran ez kihasználatlan marad, hiszen ha később, az implementálásnál használt programozási nyelv csak az egyszeres öröklődést engedélyezi, akkor ez az UML ábráin sem jelenhet meg.

Az öröklődéshez is köthető igen fontos fogalma az OO világnak az **absztrakt osztály**. Lényege, hogy az absztraktnak jelölt osztályunk nem példányosítható, viszont alosztályai lehetnek. Ábrázolásuk az UML-ben hasonló a többi osztályhoz, viszont a nevüket dőlt betűvel írjuk. Egy másik gyakran használt jelölésmód az, hogy az osztályt az <<abstract>> sztereotípiával látjuk el. Az AutoWorld osztályai közül kettő absztrakt. Mégpedig a Felhasználó és az Autó osztályok. A 7. ábrán is látható, hogy neveik dőlt betűvel vannak megadva.

A teljes elemzési osztálydiagram a 9. ábrán látható.



9. ábra Az AutoWorld alkalmazás elemzési osztály diagramja

4.1.2 Tervezési osztálydiagram

A tervezési fázisban is jelentős szerepe van az osztálydiagramoknak. Azonban itt már nem csak a fogalmakat ábrázoljuk, hanem a tervezés kibővül technikai leírással is. Az elemzési fázis ugyanis csak a szakmai szempontokat veszi figyelembe, míg tervezési szinten a technikai szempontok is megjelennek. Úgy is mondhatnánk, hogy az elemzés során a „mit" kerül tárgyalásra, míg a tervezés során a „hogyan". Tervezési szinten osztályaink részletesebb vannak megadva és jobban közelítik a megvalósítást.

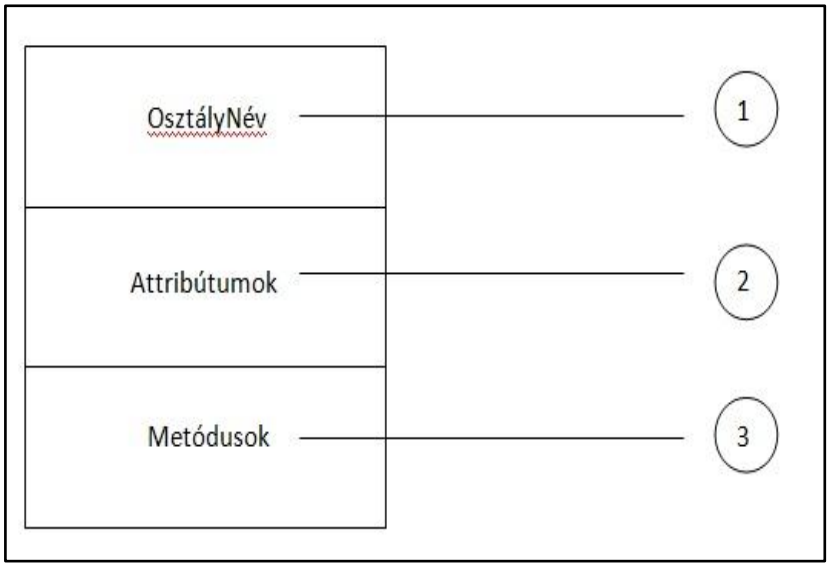
Attribútumok és metódusok

Az osztályok egyik fő alapelemei az attribútumok, amelyek különböző tulajdonságokat írnak le. Az osztályok másik igen meghatározó alapelemei a metódusok, amelyek azon viselkedést írják le, amellyel majd az osztály egy példánya fog rendelkezni. Az attribútumok és a metódusok az elemzési szinten is megjeleníthetőek. Az attribútumok ott a nevükkel és a típusukkal jelenhetnek meg csupán, míg tervezési szinten sok más információval bővülnek ki. Hasonlóan a metódusok is elemzési szinten csak a nevükkel jelennek meg, azonban a tervezési szinten további információk kerülnek megjelenítésre. Mint az kiderült az előző néhány sorból, az attribútumokat és a metódusokat külön kellett volna az elemzési osztálydiagram vonatkozásában is tárgyalni. Úgy gondoltam azonban, hogy az itt betöltött nagyobb szerepük miatt, és mivel itt sokkal több információt hordoznak, ebben a fejezetben mutatom be a rájuk vonatkozó UML jelöléseket.

Attribútumok és metódusok elhelyezkedése

Azt már korábban említettem, hogy az osztályok jelölésére az UML egy téglalapot használ, és az abban szereplő név az osztály neve. Azonban az osztályt jelölő téglalapot további részekre oszthatjuk, ahová az attribútumait és metódusait helyezhetjük el.

A felosztás a következő:



10. ábra

A részek magyarázata:

1. Az osztály neve
2. Az osztály attribútumainak listája
3. Az osztály metódusainak listája

Egy osztályt szemléltető téglalap tetszőleges számú rekeszre felosztható, amelyekben további attribútum- és metódusfajták, használati esetek, állapotautomaták, stb. jelenhetnek meg. A sorrend nem kötelezően előírt, de elfogadott szokás, hogy először az attribútumok, majd a metódusok következnek, és ezután jönnek az esetleges egyéb részek. Az AutoWorld osztályai közül emeljük ki a Foglalás osztályt. Az említett osztályt a 11. ábra mutatja. Látható az ábrán az osztályt reprezentáló téglalap hármas felosztása.

Attribútumok

Az attribútumok a következő alakban jelenhetnek meg:

[Láthatóság] [/] Név [:Típus] [Érték]

Az attribútumokat a hozzájuk rendelt nevükkel jelennek meg, amelyet értelemszerűen kötelező megadni, azonban a többi rész megadása opcionális. A név és minősítés segítségével minden attribútum egyértelműen azonosítható. Az attribútum egy fontos meghatározója a típusa. A típus a nevet követi és azt egy kettőspont választja el tőle. Típusként megadható egy az UML által ismert típus, vagy egy osztály. A Foglalás osztály attribútumai ennek megfelelően vannak megadva. A „foglalásAzonosító” nevű attribútum ezek szerint ’int’ típusú, ugyanúgy, mint a „kiFoglalt” és a „mitFoglalt” attribútumok is. A „mikorFoglalt” attribútum típusa viszont Date.

Az attribútumnak a típusa tartományából származó kezdőértéket is adhatunk. A kezdőérték megadása után a readOnly megszorítás megadása után az attribútum értéke a későbbiek során már nem lesz módosítható.

11. ábra A Foglalás osztály tervezési szinten

Az objektumorientált világban lehetőség van arra, hogy attribútumainkat elrejtsük a külvilág elől. Ehhez az UML bevezeti a láthatóság fogalmát. Tervezési szinten az osztályok egymásba ágyazott névterek, melyek határai a láthatóságnak. Az UML a következő láthatósági szinteket különbözteti meg:

+ : Az attribútum az egész névtérben látható.
# : Az attribútum minden olyan elemben látszik, amely általánosítása azon osztálynak, amelyben az attribútum definiálva lett.
~ : Az attribútum az ugyanabban a csomagban található elemek számára látható.

− : Az attribútum csak abban a névtérben látszik, amelyben definiálva lett (általában a definiáló osztály).

Az UML láthatósági szintjei nagyjából megfelelnek a Java láthatóságainak, tehát rendre a public, protected, package és private láthatósági szinteknek. A láthatóság jelét az attribútum elé írjuk. A láthatóság megszorításként is megadható, például a {protected} alakban. A Foglalás osztály összes attribútuma private láthatósággal van deklarálva.

Az attribútumok neve előtt közvetlenül megadható egy törtjel. Ezzel származtatott attribútum definiálható. Származtatott attribútum lehet öröklött, vagy más attribútumokból kiszámított, úgynevezett tranziens attribútum. Osztályaink tartalmazhatnak olyan attribútumokat, melyek nem az egyes példányokhoz tartoznak, hanem a definiáló osztályhoz. Ezeket osztályattribútumoknak nevezzük, és az UML jelrendszere szerint a neve előtti aláhúzással jelöljük. Ez a jelölés a Java-ban és C++-ban ismert static kulcsszavas deklarálás megfelelője.

Metódusok

Az UML-ben egy metódus deklarációja a következő lehet:

[Láthatóság] Név ([ParaméterLista]) [:Típus] [{ Tulajdonságok }]

A metódusukhoz megadható láthatósági szintek szintaktikailag és jelentésileg is teljes mértékben megegyeznek az attribútumoknál tárgyalt láthatósági szintekkel. Az UML ajánlása szerint a metódusok neveit kisbetűvel kell kezdeni. Egy művelet nem azonosítható egyértelműen csak a neve alapján. Ugyanis mint sok programozási nyelvben, az UML-ben is létezik a műveletek túlterhelésének fogalma. Ennek lényege, hogy azonos nevű metódusok különbözőek lehetnek, ha a paraméterlistájuk vagy visszatérési értékük különbözőek. Ezért egy metódust egyértelműen csak a szignatúrája alapján tudunk megkülönböztetni, amely a névből és a paraméterlistából, valamint a visszatérési típusból épül fel. A paraméterlista állhat egy paraméterből, de állhat véges sok paraméterből is. Több paraméter megadása során a paramétereket vesszővel kell elválasztani. Ugyanakkor előfordulhat, hogy a paraméterlista üres, akkor a zárójelek között nem szerepel semmi.

Egy paraméter felépítése a következő:

[Irány] Név : Típus [=Érték]

Paraméter esetében is csupán a név megadása kötelező az UML-ben. A név után kettőspontot követően megadható a paraméter típusa, amely meghatározza a paraméter által felvehető értékek tartományát. A típus után megadható egy alapértelmezett érték, amely akkor kap értelmet, ha a megfelelő paraméter nem kap aktuális paramétert. A paraméter megadásában szerepel egy Irány rész, amely opcionális rész. Ebben a részben a paraméterátadás módját adhatjuk meg. Az alapértelmezett paraméterátadási mód az „in" módú átadás, ezért csak akkor kell külön átadási módot megadni, ha másikat szeretnénk használni. A következő paraméterátadási módok közül választhatunk az UML szerint:

in :	a paraméter értéke felhasználható, de nem változtatható meg
out :	a paraméter értéke módosítható a metódus által
inout :	a paraméter értéke felhasználható és írható is
return :	a paraméter a metódus visszatérési értéke

A metódus deklarációjában, a paraméterlistát követően megadható a metódus visszatérési értékének típusa. Ha nem szerepel típus, akkor a metódus egy eljárás lesz. A Foglalás osztály metódusai közül, ha kiemeljük a „setMitFoglalt" metódust, látható hogy publikus láthatóságú, egy paramétert vár, amely Date típusú, és nincs visszatérési értéke, amit a „void" szó jelez.

Szintén kitüntetett szerepük van a metódusok körében a lekérdező és a beállító metódusoknak. Egy lekérdező metódus visszatérési értéke egy attribútum értéke, míg egy beállító metódus a paraméterének kapott értéket állítja be az attribútum értékének. E két metódusfajta segítségével elrejtjük az attribútumainkat, amelyek csak ezen metódusokon keresztül érhetőek el. A könnyebb azonosíthatóság érdekében a lekérdező metódusok neveit az attribútum nevéből képezzük úgy, hogy a név elé egy „get" prefixet írunk. Abban az esetben, amikor az attribútum logikai típusú akkor a prefix „is" lesz. A beállító metódusok esetében hasonlóan járunk el, csak ott a prefix egységesen „set" lesz. A „get" és „set" prefix a metódusfajták angol elnevezéséből adódik (lekérdező metódus – getting method; beállító metódus – setting method). A két metódusfajta az osztályokban külön részben, csoportosítva,

a << property get>> és a <<property set>> sztereotípiákkal felvezetve is megjeleníthető, amely a 11. ábrán látható osztály valamennyi metódusára igaz.

Egy osztály tartalmazhat olyan metódusokat is, amelyeknek csak a specifikációját adjuk meg, a törzsüket azonban nem implementáljuk. Ezeket a metódusokat absztrakt metódusoknak nevezzük. Ha egy osztály tartalmaz egy absztrakt metódust, akkor maga az osztály is absztrakt lesz. Az UML szerint az absztrakt metódusokat az { abstract } megszorítással jelezzük, vagy úgy, hogy dőlt betűvel írjuk őket.

Metódusok egy fajtája az osztály szintű metódusok, amelyek az osztályattribútumokon operálnak. Jelölésük az UML-ben annyiban különbözik a többi metódushoz képest, hogy a neveiket alá kell húzni.

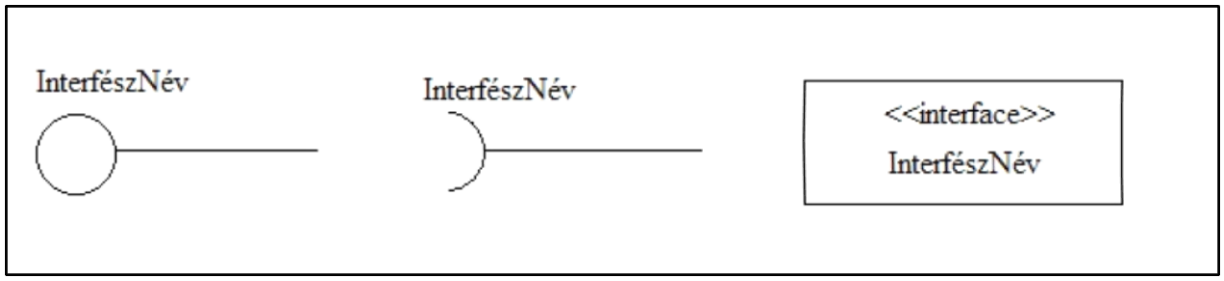
A tervezési osztálydiagram további elemei lehetnek

Az osztályok közötti kapcsolatok hasonlóképp jelennek meg itt is, mint az elemzési szinten. Hasonlóan jelöljük absztrakt osztályainkat is. Azonban itt már megjelennek az úgynevezett aktív osztályok. Aktív osztály alatt olyan osztályt értünk, amely önmagától aktivizálódik, tehát nem kell külső esemény a beindításukhoz. Ezek az osztályok az operációs rendszer folyamatainak vagy szálainak felelnek meg, illetve a főalkalmazást reprezentálják. Az aktív osztályokat jelölhetjük vastagabb keretű téglalapokkal, vagy a téglalap két szélső oldalát dupla vonallal rajzolva. Egy másik módja az aktív osztályok megadására az, hogy az osztály neve felé egy <<activeClass>> sztereotípiát illesztünk.

12. ábra Az aktív osztály három jelölési alternatívája

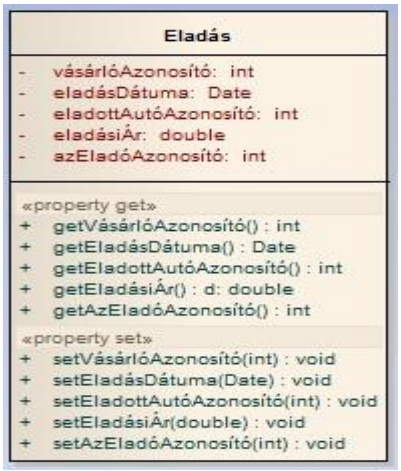
A tervezési osztálydiagram kifejezőeszközei között találunk még olyan eszközt, amelyet az elemzési osztálydiagramon nem ábrázolhatunk. Ilyen például az interfész. Az interfészek csak metódus specifikációkat tartalmazó programozási eszközök, amelyek a

metódusok törzsét nem implementálják. Tehát az interfészek kizárólag publikus, absztrakt metódusokat tartalmazhatnak, attribútumokat azonban nem, legfeljebb konstansokat. Az UML-ben az interfészeket jelölhetjük egy téglalappal is, amelyben az interfész neve felé az <<interface>> sztereotípiát helyezzük el. Másik jelölési mód a gömbcsukló, amikor a nevet e felé írjuk. Interfészek tekintetében beszélhetünk szolgáltatott interfészről, és beszélhetünk felhasznált interfészről. Az előbbi jele egy kör, az utóbbié egy félkör.



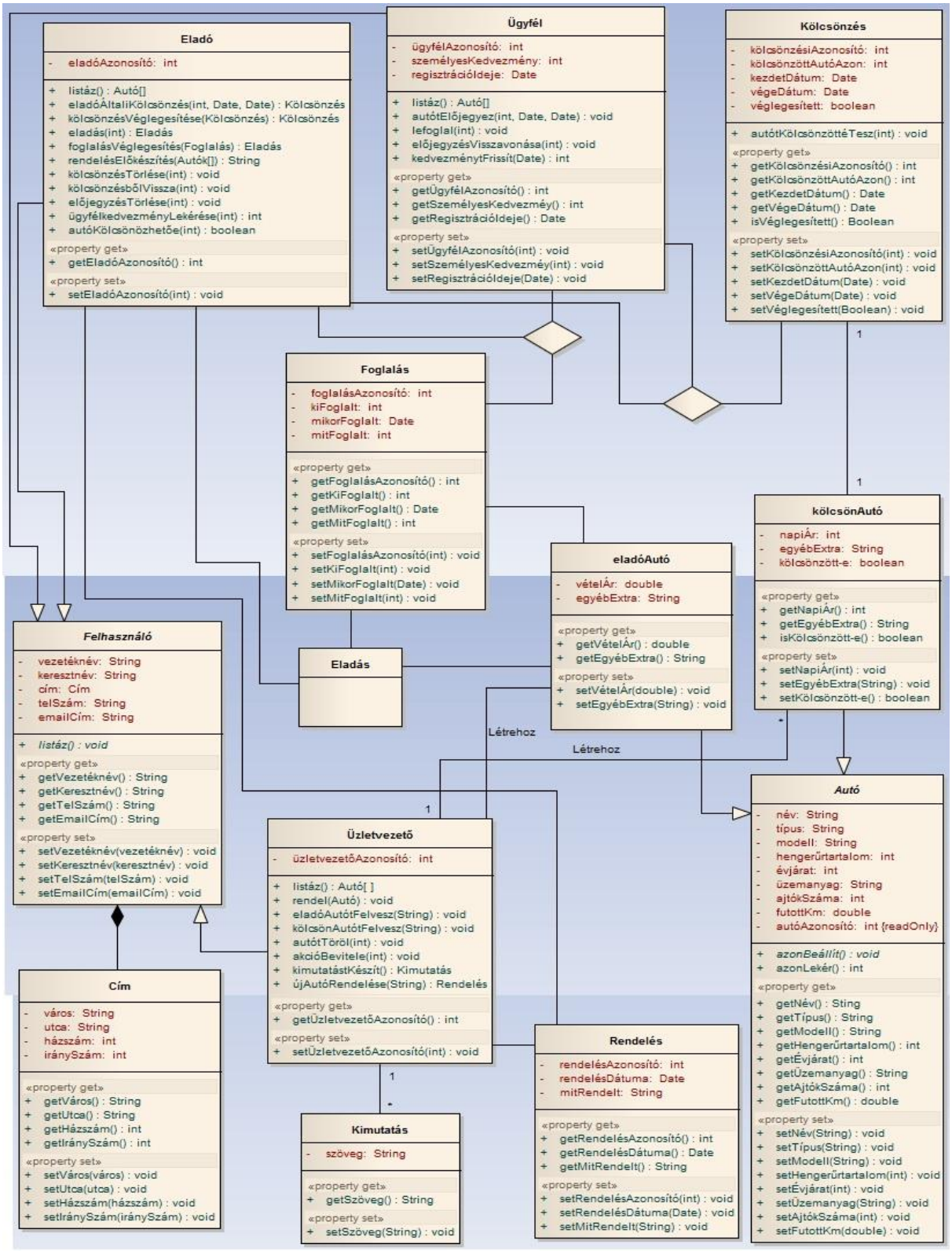
13. ábra Az interfész jelölési módjai: szolgáltatott és felhasznált interfész; az interfész sztereotípiával

Az interfésznek is lehet általánosítása és pontosítása, így köztük is fennállhat az öröklődési viszony. Az interfészek között többszörös öröklődés van. Egy osztály implementálhat egy interfész, ha az osztály megvalósítja az interfész által tartalmazott összes metódust. Tehát az osztály ekkor az egyébként absztrakt metódusok törzsét is implementálja. Azt, hogy egy osztály implementál egy interfészt, az UML egy speciális viszonnyal, a realizációval jelöli. Realizáció esetén az osztály az interfésznél megadott összes elemmel rendelkezik. A realizációt az UML egy fehér háromszögben végződő szaggatott vonallal jelzi, amely a megvalósító osztálytól a megvalósított interfészig vezet.



14. ábra: A tervezési osztálydiagramon szereplő Eladás részletezése

Az AutoWorld alkalmazáshoz készített tervezési osztálydiagram



15. ábra

4.1.3 Megvalósítási osztálydiagram

Az elemzési osztálydiagram segítségével felvázolható volt a megoldandó feladat szakterületének fontosabb fogalmai, valamint körülírható volt az, hogy mit szeretnénk megvalósítani, elérni a készülendő rendszerünkkel. A tervezési osztálydiagramon már nagyobb szerepet kapott a megvalósításhoz kapcsolódó technikai kérdések köre, azonban ezen a szinten nem beszélhetünk még konkrét megvalósítási nyelvről, nem jelennek meg a nyelv későbbi konkrét konstrukciói. Mégis a tervezési osztálydiagram szolgáltat alapot a megvalósítási osztálydiagram számára, hiszen ezen a magasabb szinten a tervezési osztálydiagram néhány elemét konkretizáljuk a használt megvalósítási nyelv eszközeivel.

A megvalósítási osztálydiagram profilja

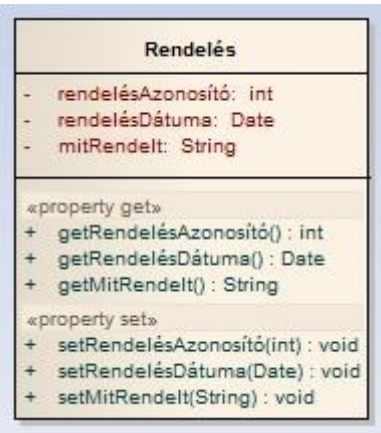
A megvalósítási szinthez tartozó osztálydiagram elkészítésénél egy profilt alkalmaznak a megvalósítási nyelvhez. Egy ilyen profilban néhány plusz kifejezőeszköz is bekerül, azonban a meglévőekre tartalmazhat korlátozást. Ha egy modell megfelel egy ilyen profilnak, akkor közvetlenül átvihető az implementációs nyelvre. Ha ez a megfeleltetés létrejöhet, akkor a munka modellvezérelt módon folytatódhat, ugyanis a változásokat elegendő a modellben elvégezni, mivel a fordítás során előáll a modellből a futtatható rendszer.

A Java, mint megvalósítási nyelv

A kellő megszorítások mellett egy megvalósítási osztálydiagram értelmezhető egy Java programként. Az Osztály class, az absztrakt osztály pedig abstract class lesz, míg az interfészek interface-k, a csomag pedig package lesz. Az attribútumok és a bináris, egy irányban navigálható asszociációk példányváltozóként jelennek meg. Változatlan formában marad az elemek láthatósága és típusa, ha típusként a Java alaptípusait, könyvtári osztályait vagy a saját osztályainkat használtuk. Az általánosítást az <<extends>>, az interfészek implementálását a <<reliaze>>, míg az importálást az <<includes>> sztereotípiával jelöljük.

Az UML-ben a static, main, final Java elemeknek nincs megfelelője, ezért ezeket a nevükből képzett sztereotípiával kell jelölni. UML-ben lévő osztályaink neve felett sztereotípiával jelölhetjük, hogy ez az osztály megfelel egy Java class-nak.

Végül tekintsük meg, hogy egy UML osztálynak milyen Java kód felelne meg. Legyen a szóban forgó osztály a következő (16. ábra) és a hozzá tartozó kód (17. ábra):



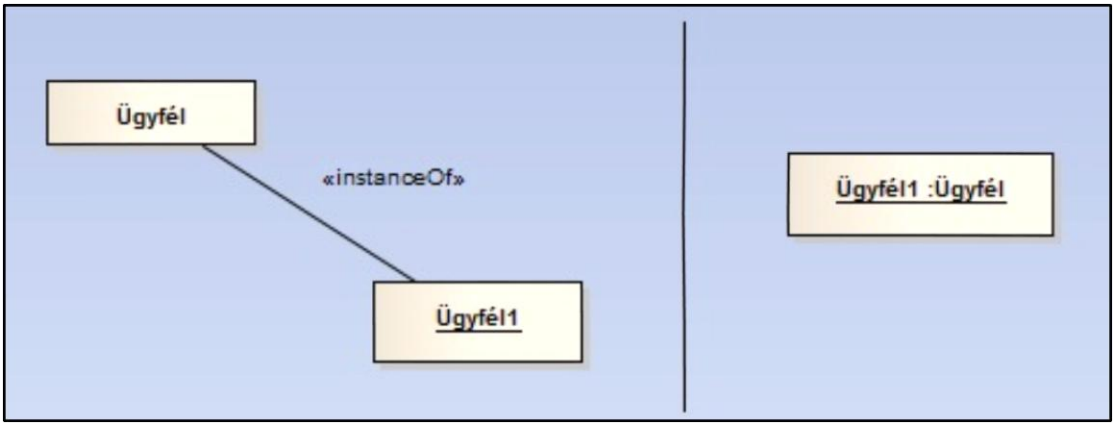
16. ábra

```
public Class Rendelés{
        private int rendelésAzonosító;
        private Date rendelésDátuma;
        private String mitRendelt;
        public int getRendelésAzonosító (){
                return rendelésAzonosító;
        }
        public void setRendelésAzonosító (int s){
                rendelésAzonosító = s;
        }
        public Date getRendelésDátuma (){
                return rendelésDátuma;
        }
        public void setRendelésDátuma (Date d){
                rendelésDátuma = d;
        }
        public String getMitRendelt (){
                return mitRendelt;
        }
        public void setMitRendelt (String s){
                mitRendelt = s;
        }
}
```

17. ábra

4.2 Objektumdiagram

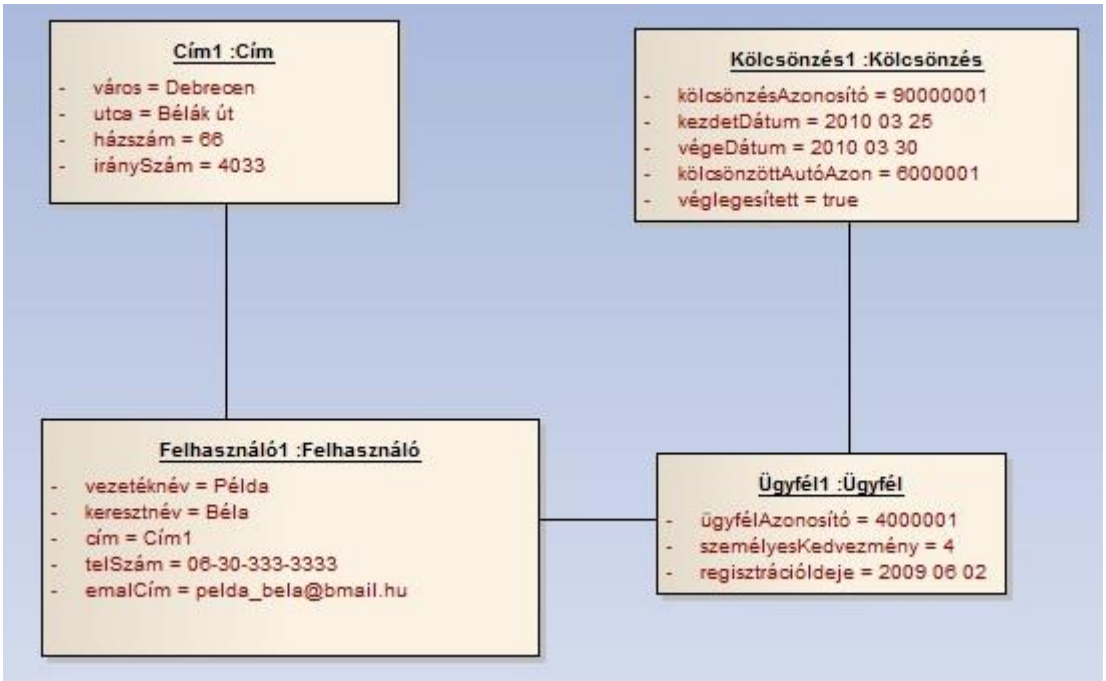
A fejlesztés során előfordulhat olyan eset, amikor az elkészülendő rendszer egy későbbi, adott idejű állapotát szükséges már abban az adott fázisban megjeleníteni, esetleg kielemezni. Előfordulhat, hogy erre egy teszteset készítésekor van szükség; de az is, hogy egy kiemelt fontosságú állapotot hangsúlyozzunk, vagy pont egy nem kívánt állapot elkerülésére kell felhívni a figyelmet. Egy rendszer adott pillanatban vett állapotát objektumainak állapotai tükrözi. Egy objektumdiagram tartalmazza a rendszer egy adott pillanatában vett konkrét objektumokat, és a közöttük fennálló kapcsolatokat. Az osztálydiagram osztályaihoz hasonlóan az objektumok is téglalap formájában jelennek meg az objektumdiagramon, ahol megadható az objektum neve, és megadható annak az osztálynak a neve is, amelyből példányosítva lett. A két név egyszerre is megadható; ekkor az objektum név áll elől, és ezt követi az osztály neve, amelyet közvetlenül egy kettőspont előz meg. Az osztályok és objektumok megkülönböztethetősége céljából az UML szerint az objektum azonosítására szolgáló neveket alá kell húzni. Az objektumdiagramon szerepelhet az objektumhoz tartozó osztály is, és ekkor a kapcsolatuk egy <<instanceOf>> sztereotípiával megadott függőséggel is jelölhető, amely az osztály felé mutat. A 18. ábra egy ügyfél objektum jelölését mutatja be.



18. ábra Két lehetséges megadása az osztálya-példánya kapcsolatnak

A modellezett valóság tulajdonságai elsősorban az osztályok attribútumaival írható le. Azonban ezen attribútumok nem tartalmaznak értéket, csupán definíciók. Az osztályok példányaiként létrejövő objektumok azonban ezen attribútumokat konkrét értékkel töltik fel.

Természetesen az objektumdiagramon így megadhatóak az objektumokhoz kapcsolódó attribútumok aktuális értékei. Általában nem az összes attribútum jelenik meg a diagramon, csupán azok, amelyek az adott mondanivaló alapján lényegesek. Egy attribútum értéke az attribútum neve és egy egyenlőség után adható meg. Az attribútum neve és egy kettőspont után megadható az attribútum típusa is, ha azt ki szeretnénk hangsúlyozni. Ha az attribútum értéke összetett, akkor azt kerek záró jelek között adhatjuk meg. A 19. ábra egy konkrét felhasználót, pontosabban egy ügyfelet ábrázol, akihez tartozik egy véglegesített kölcsönzés.

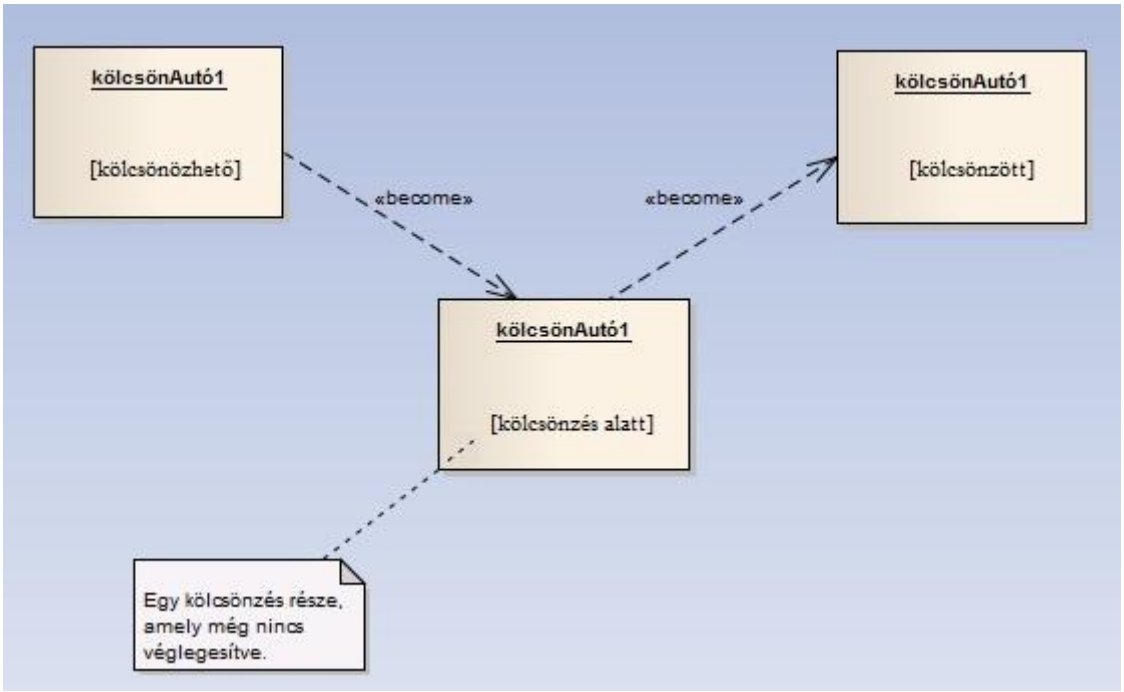


19. ábra

A fent látható diagramrészlet egy Ügyfelet, nevezetesen „Példa Bélát” mutatja be. A diagram közel sem teljes, hiszen nem szerepel rajta az AutoWorld rendszerben adott pillanatban létezhető összes objektum, azonban ez elfogadható és nem okoz hibát. Ahogy pár sorral fentebb már említettem, elég csupán azt a részét ábrázolni a rendszernek, amely egy bizonyos indok miatt épp akkor fontos. Tegyük fel tehát, hogy ez az objektumdiagram, csak egy Ügyfél objektum megjelenését hivatott bemutatni a rendszerben, a hozzá kapcsolódó kölcsönzés objektummal együtt. Ha az objektumdiagram elkészültének ez volt a tényleges oka, akkor megfelelő diagramot sikerült előállítani. A diagramon látható, hogy az objektumok neveit és típusait is aláhúztam, közéjük pedig kettőspontot tettem, így jelölve objektum voltukat. Az

objektumok attribútumai maradéktalanul ábrázolva lettek, de nem lett volna az összesre szükség. Az attribútumoknál meg lettek adva azok nevei és egyenlőség jel után a konkrét értékek. Itt még megadható lett volna az attribútumok nevei után azok típusai is. Az objektumok között fennálló kapcsolatokat asszociációk jelölik.

Bár egy objektum állapotát attribútumai határozzák meg, a diagramon jelölhető az állapot szövegesen, szögletes zárójelek között a név alatt. Ha egy objektum állapotának időbeli változása megadható az egyes fázisok elnevezésével, akkor a változatok összekapcsolása, a <<become>> sztereotípiával jelölt függőségekkel kitűnően szemléltetheti a változásokat. A szalon által kölcsönzésre kínált autók is több állapotban lehetnek. Az autó alapállapota a „kölcsönözhető”, majd lehet egy kölcsönzés része, amely még nincs véglegesítve, legvégül kölcsönzötté válik. Ezt mutatja be a 20. ábra.



20. ábra Példa a <<become>> sztereotípia használatára

Két objektum kommunikálhat egymással, elérheti egyik a másikat. Ezeket a kapcsolatokat az objektumokat összekötő vonallal jelöli az UML. Az objektumok a köztük fennálló kapcsolatok lévén műveleteik során hatással lehetnek más objektumokra. Ha egy objektum műveletének egy másik objektumra vonatkozó hatását szeretnék megjeleníteni, akkor az ábrázolt kapcsolat megfelelő oldalán jelölhető ez a hatás a megfelelő megszorítással.

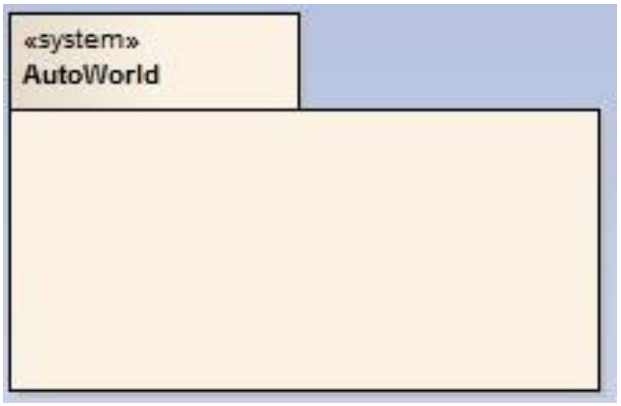
Ezek a következőek lehetnek:

- {new} : A kapcsolat másik oldalán lévő objektum a megadott művelet során jön létre.
- {destroyed} : A művelet során a túloldali objektum megszűnik.
- {transient} : A túloldali objektum élettartama megegyezik a jelölt művelet futási idejével.

Olyan eset is előfordulhat, hogy megvizsgáljuk egy adott pillanat aktív és passzív objektumait. Aktív objektum rendelkezik a vezérléssel, míg a passzív objektum csak megkaphatja a vezérlést egy hívás során. Az aktív objektumok megkülönböztetésére az UML a vastagított oldalú téglalapot ajánlja, de használhatjuk az {active} megszorítást is.

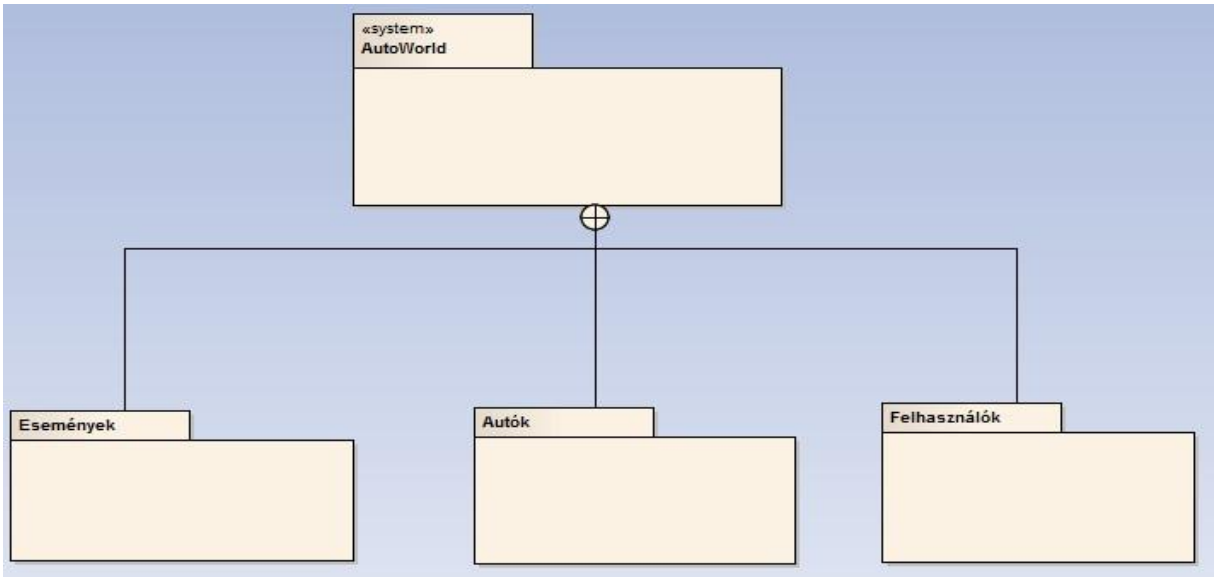
4.3 Csomagdiagram

Csomagdiagramok segítségével a modellezett rendszer fontosabb elemei csoportosíthatóak egy adott szempont szerint. A csoportosítás révén a megfelelő elemek közös névtérbe kerülnek. A csomagokra bontás által az alkalmazás részei áttekinthetőbbé válnak. A csomagok egymásba ágyazhatóak. A legkülső csomag kivételével minden további csomag csak egy (másik) csomagba lehet beágyazva. Ennek következtében kialakul egy fa struktúra a csomagok között.

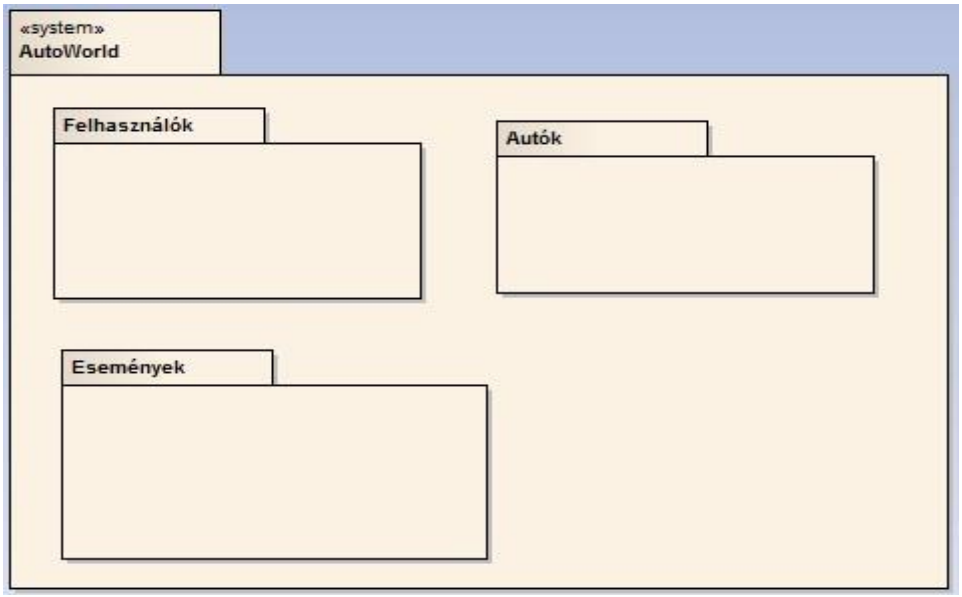


21. ábra Az AutoWorld csomag

Megkülönböztetett szerepű a legkülső csomag, amely a teljes alkalmazást tartalmazza, és általában a <<system>> sztereotípiával van jelölve jelölni. Az AutoWorld esetében a 20. ábrán látható a külső csomag, amelynek neve megegyezik a rendszer nevével. Látható, hogy a csomag egy címkézett téglalappal van jelölve, és a címkébe került a név. A csomag neve megadható lett volna a csomagot ábrázoló elem közepébe írva is. Egy csomag több csomagot is tartalmazhat, amelyeket a csomag belsejében kell elhelyezni hasonló jelöléssel. Ennek ábrázolására kétféle módszer is létezik, amelyet a 22. és 23. ábra mutat be.

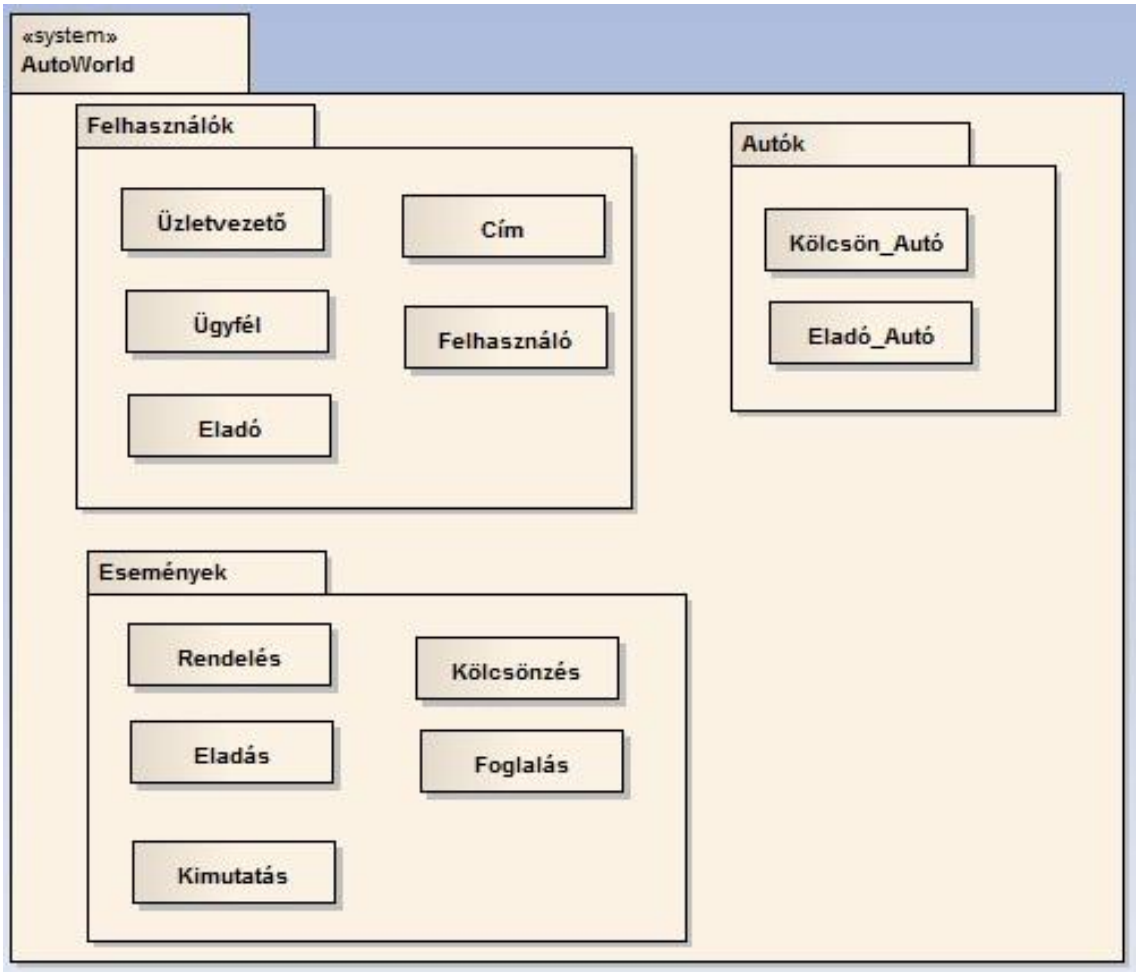


22. ábra



23. ábra

A továbbiakban a 23. ábra jelölési módszerét használom majd. Az ábrák alapján látható tehát, hogy az AutoWorld rendszer elemeit három csoportba lehet besorolni: Felhasználók, Autók és Események. A csomagok tartalma a 22. ábrán és a 23. ábrán grafikusan lett megadva, azonban a tartalom megadható lett volna felsorolva, szöveges formában is. Az AutoWorld esetében a teljes rendszer a 24. ábrán látható.



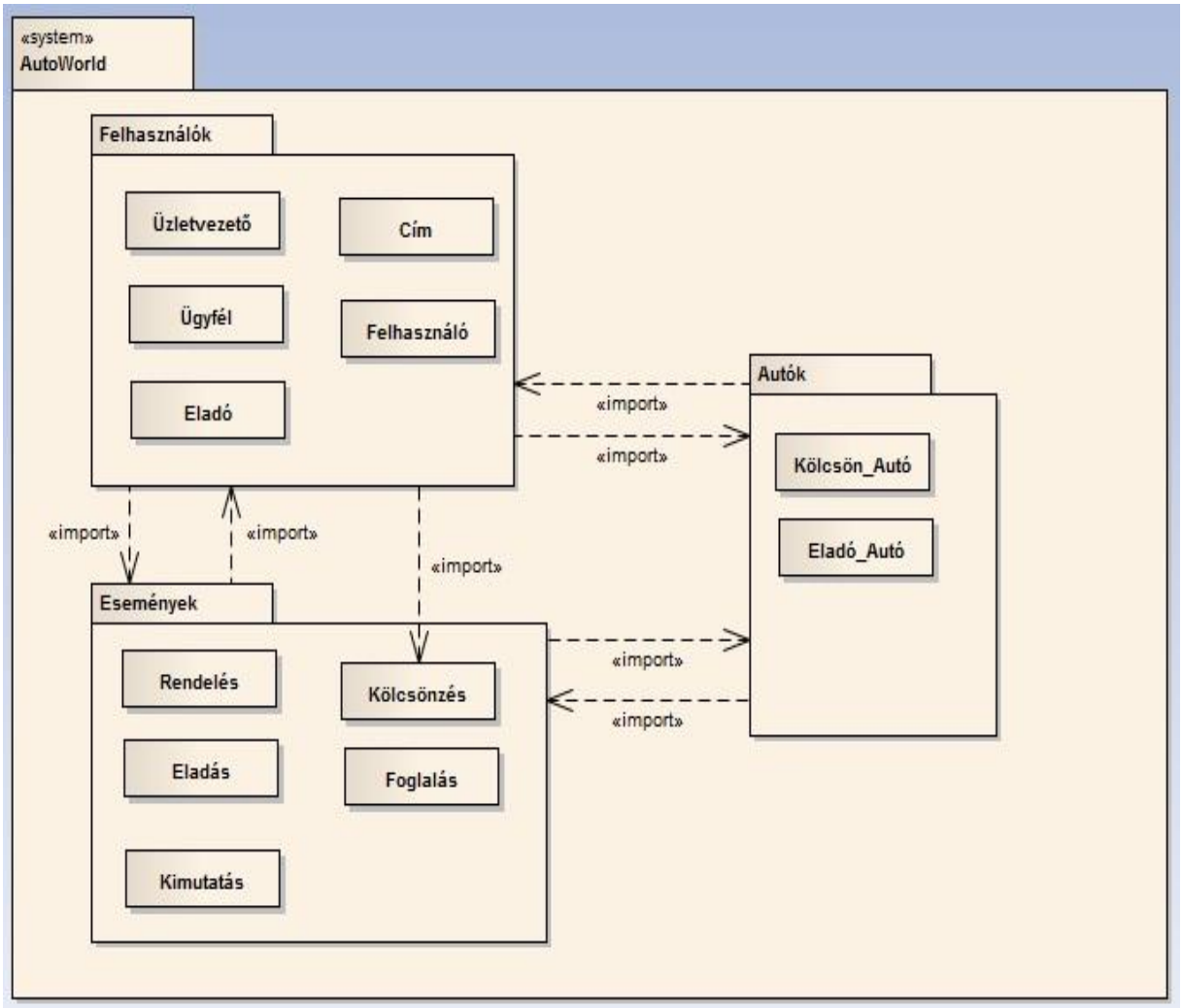
24. ábra A rendszert alkotó csomagok

Mint ahogy már említettem a csomag egy névtér is egyben, ebből kifolyólag az egy csomagon belül lévő elemek közvetlenül hivatkozhatnak egymásra. Azonban ennek biztosítása érdekében a csomagon belül, adott szinten minden névnek egyedinek kell lennie, a nem egy szinten álló csomagok elemit pedig minősített névvel lehet elérni. Egy más csomagban lévő elemre való hivatkozáshoz meg kell adni a hozzá tartozó elérési utat, amelyet

szintén minősítéssel adhatunk meg. Például, ha az Eladó_Autó-t kell hivatkozni, akkor a következőképp lehet elérni:

AutoWorld::Autók::Eladó_Autó

Mint az látható az UML a minősítés operátorának a dupla kettőspontot használja. A teljes elérési utat tartalmazó hivatkozás kezelése nagyon-nagy rendszereknél igen bonyolult ás fárasztó lehet. Ezért az UML bevezette a függőségek egy külön csoportját, az importot. Import segítségével elérhetjük a kívánt elemet. Ábrázolásához szaggatott, nyitott hegyű nyilat használunk.



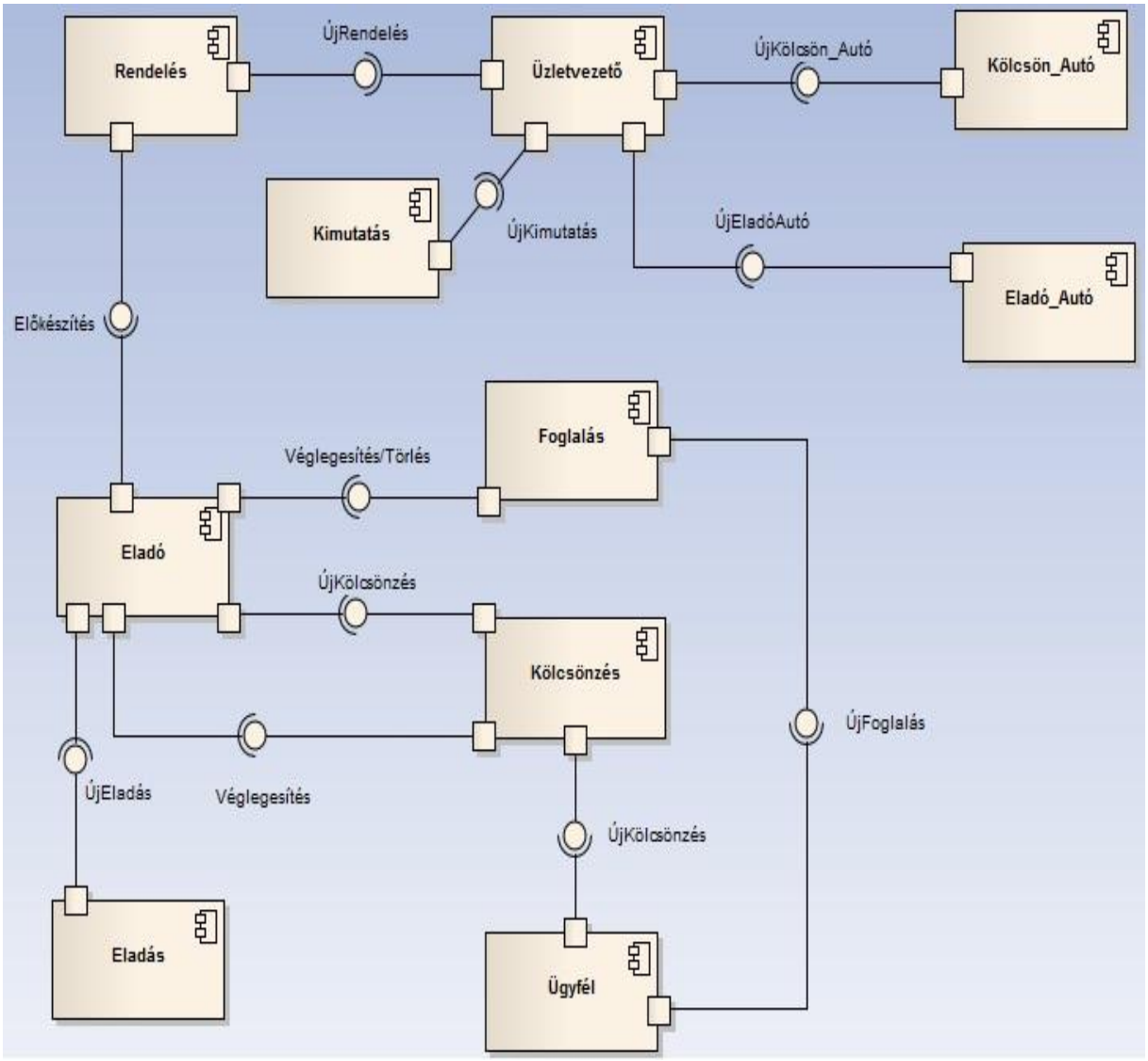
25. ábra A teljes csomagdiagram

A 25. ábra már az importálásokkal együtt tartalmazza a csomagokat. Az import kapcsolatot jelző nyíl mindig az importálótól, az exportáló felé tart. Kétféle import szerepel az ábrán: egyedi import és csomagimport. Egyedi import kapcsolat áll fenn Felhasználók csomag és az Események csomag Kölcsönzés eleme között, ekkor az importot jelző nyíl a csomag felől az elemig tart. Az összes többi import csomagimport.
A csomagbeli elemekhez láthatóság is rendelhető. Azonban csak a public és a private adható meg. A 25. ábrán szereplő csomagbeli elemek mind nyilvános láthatóságúak, ezt külön nem kellett jelölni.

4.4 Komponensdiagram

Komponensdiagram segítségével a rendszer komponenseit és a köztük lévő kapcsolatokat tudjuk szemléltetni. Egy rendszer komponensekre való osztása azonban sokféle lehet. Ez abból adódik, hogy az UML komponens-definíciója igen széles határokat hagy afelől, hogy mi is lehet komponens. Ugyanis az UML szerint komponens az, ami egy egységbezárt, önálló, teljes és cserélhető egység, amely függetlenül működtethető, telepíthető és összekapcsolható más komponensekkel. A komponens megjelenhet tehát a szoftveréletciklus csaknem összes fázisában, más-más szerepben. Jelölhető komponenssel logikai és fizikai egység egyaránt.

Az 26. ábra az AutoWorld rendszerhez tartozó egy komponensdiagramot ábrázol. A diagramon a rendszer futása közben megjelenő egységek vannak megjelenítve. Az UML a komponens jelölésére egy téglalapot használ, melynek jobb felső sarkában egy a 26. ábra komponensein is látható jelet helyez el. Azonban a komponens ábrázolható lenne úgy is, mint osztály, egy <<Component>> sztereotípiával. Egy komponens rendelkezhet interfészekkel, amelyek lehetnek elvárt interfészek, mint például az Eladó komponens interfészei, és lehetnek szolgáltatott interfészek is, mint például az Eladás interfésze. A komponenseknek lehetnek csatlakozói (portjai), amelyek kis négyzettel jelöl az UML. Egy csatlakozó a komponens összes interakcióját összefogja, a nyújtott és az elvárt interfészeket, sőt ezek használati protokollját is. A komponensek maguk is tartalmazhatnak további komponenseket.

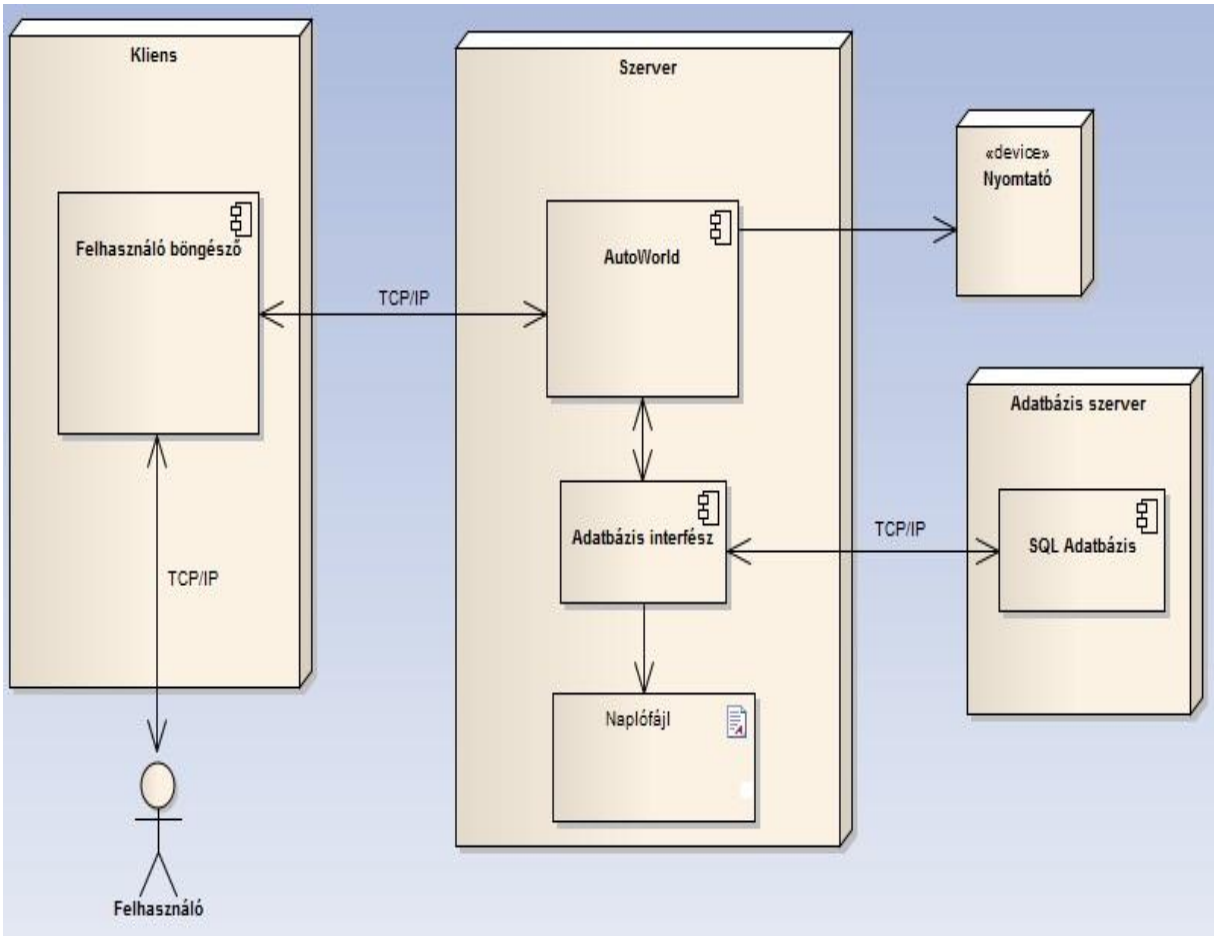


26. ábra: A komponensdiagram

Ha egy komponens összetett, akkor a külvilág és a tartalmazott komponensek közötti kapcsolatot a delegáló összekötő adja. A delegáló összekötő a komponens szolgáltatásának hívását közvetíti a komponens részei felé, és lehetőséget biztosít a részeknek arra, hogy teljes komponensként hívják meg más külső komponens szolgáltatását. Jelölésére az UML a <<delegate>> sztereotípiát használja.

4.5 Kihelyezési diagram

A kihelyezési diagram segítségével ábrázolhatóak azok a hardveres egységek és a köztük fennálló fizikai kapcsolatokat, amelyeken majd a kész alkalmazás futni fog. A diagram segítségével definiálhatjuk, hogy az alkalmazás milyen készülékek csatlakozását igényli.



27. ábra: Az AutoWorld kihelyezési diagramja

Az AutoWorld rendszerhez tartozó kihelyezési diagram a 27. ábrán látható. A kihelyezési diagram alapeleme a csomópont, amelyet téglatesttel jelöl az UML. A diagramon tehát csomópont a kliens, a szerver, az adatbázis szerver és a nyomtató. Mint látható a Nyomtató mint csomópont egy <<device>> sztereotípiával van ellátva, amely jelzi, hogy a csomópont számítási kapacitással nem rendelkezik. Az egyes csomópontok további kisebb

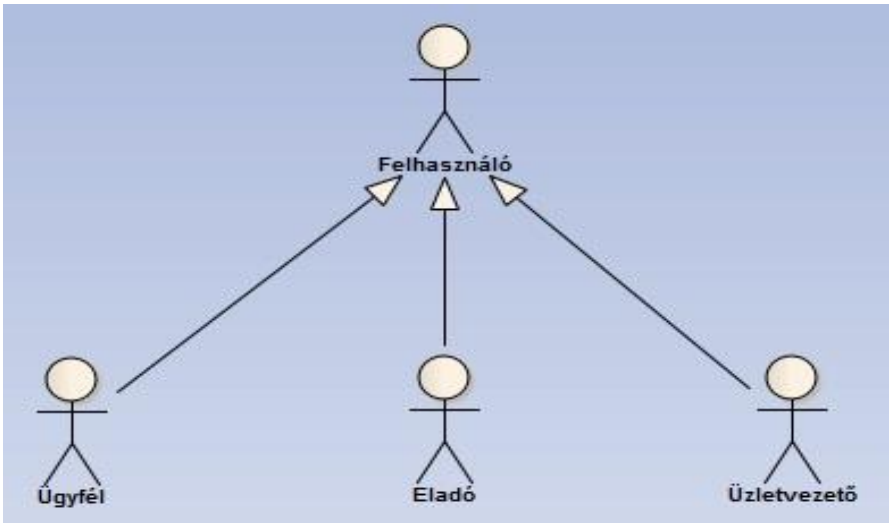
egységekből épülnek fel, amelyek azonban már nem hardver egységek. Ezen egységek komponensekként jelennek meg a csomópontokon. Természetesen egy csomópont a feltüntetett komponenseken kívül további elemeket is tartalmazhatnak, azonban elegendő azokat ábrázolni, amelyek az alkalmazásunk tekintetében fontosak. A Kliens csomópont tartalmaz egy Felhasználói böngészőt; ezen keresztül tud kapcsolatba kerülni az AutoWorld felhasználója az alkalmazással, akit a diagramon a Felhasználó nevű actor jelöl. Az AutoWorld alkalmazásunk egy szerveren fut majd. Az alkalmazás egy adatbázisban tárolja adatait, amely külön csomópontként van jelölve. A Szerveren belül található még egy Adatbázis interfész komponens, amely az Adatbázis szerverrel való kommunikációért felelős. Ehhez a kommunikációhoz egy naplózási fájl is tartozik. A napló fájl egy fizikailag is megjelenő fájl, amelyet az őt jelölő téglalap jobb felső sarkában elhelyezett dokumentumra hasonlító jel mutat. Az egyes csomópontok, vagy a csomópontok által tartalmazott komponensek kommunikálhatnak más csomópontokkal, vagy komponensekkel, amely nyíllal van jelölve. A nyíl egyértelműsíti az adatforgalom irányát. A nyilak neveként megadhatjuk a kommunikációs kapcsolat fajtáját, amely az AutoWorld esetében TCP/IP kapcsolatokat jelent.

5. Viselkedési diagramok

5.1 Használati eset diagram

A használati eset (angolul use case) technikát Jacobson és társai fejlesztették ki. Céljuk a későbbi rendszerrel támasztott követelmények összegyűjtése és áttekinthető ábrázolása volt. Ugyanis a használati esetek előtt a követelményeket csupán szöveges dokumentumok tartalmazták. A használati esetek segítségével könnyen ábrázolhatjuk az alkalmazáshoz kapcsolódó külső szereplőket, valamint azt, hogy milyen funkciókon keresztül csatlakoznak ezek a szereplők az alkalmazáshoz. A diagram segítségével rögzíthetjük a felhasználók egyes követelményeit, tehát azt, hogy mit szeretnének majd az alkalmazás segítségével végrehajtani. Azonban ennek tényleges megvalósításáról nem esik szó, kizárólag a kívánt funkciók felsorolása történik meg.

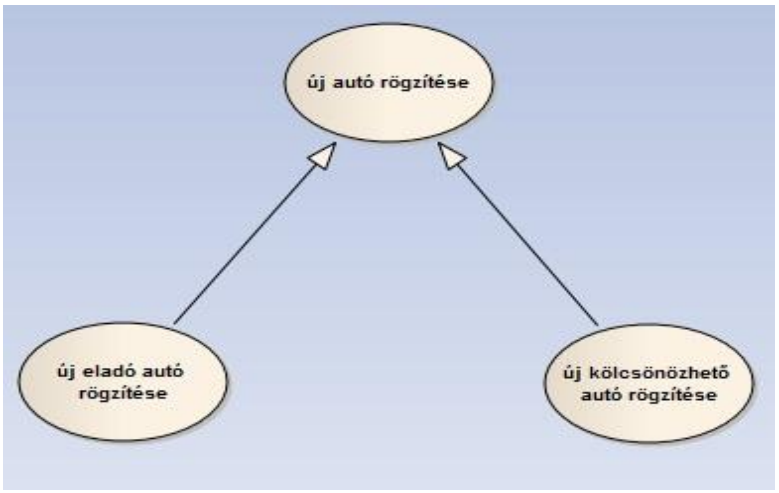
A kapcsolódó elemeket aktoroknak nevezzük. Aktor lehet az alkalmazást használó személy is, de lehet egy a rendszeren kívüli alrendszer is. Az AutoWorld esetében aktorok lehetnek a Felhasználók; azon személyek tehát, akik használni fogják majd a rendszert.



28. ábra: Az AutoWorld aktorjai

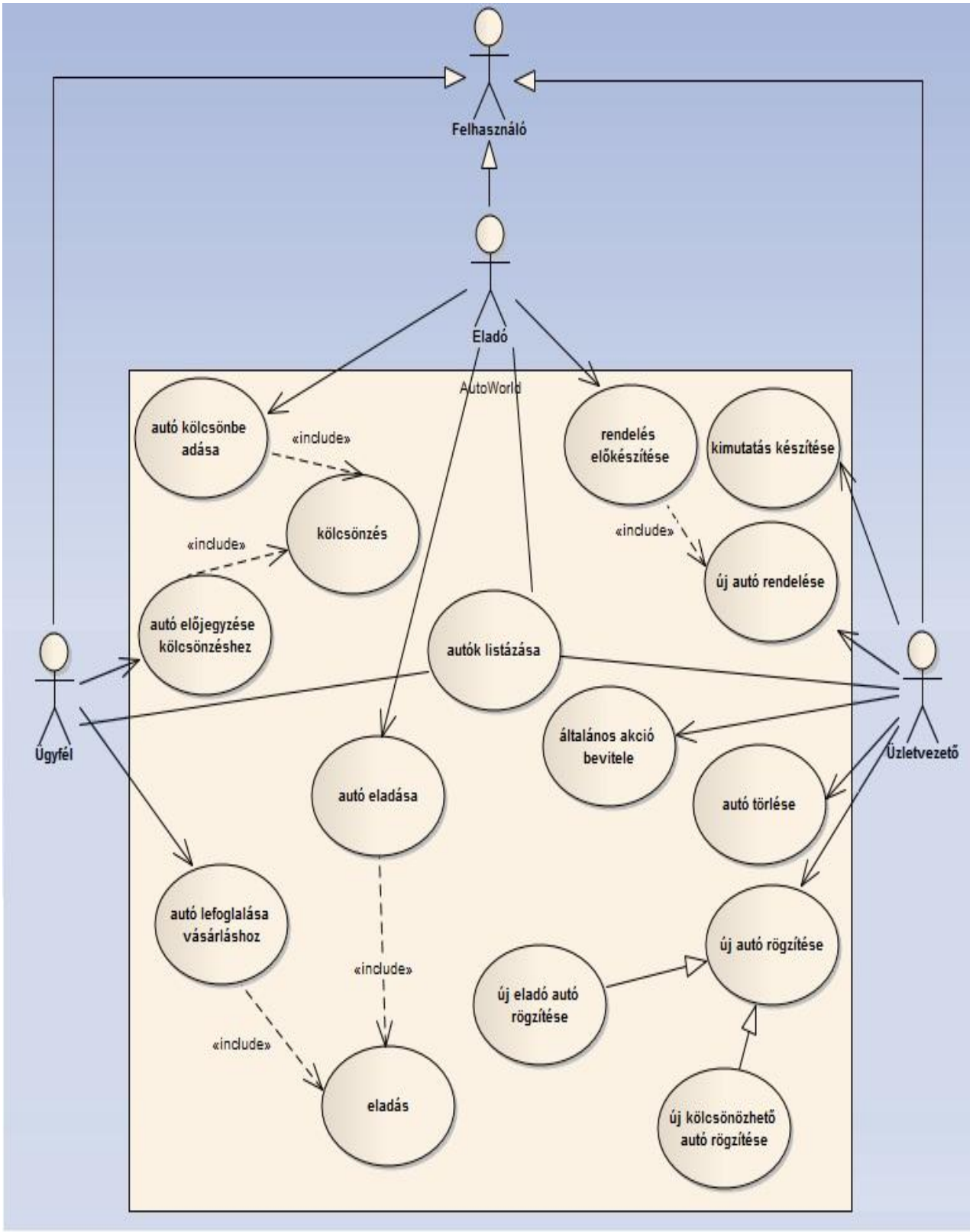
A 28. ábra az AutoWorld-höz tartozó aktorokat mutatja be. A Felhasználó az AutoWorld esetében három speciálisabb személy lehet. Ezek az Ügyfél, az Eladó és az Üzletvezető. Ezek is aktorok. Az aktorokat mint az a 28. ábrán is látható úgynevezett pálcikaemberekkel lehet jelölni. Másik jelölés lehet még, hogy ha olyan osztályként jelöljük az aktorokat, amelyek <<actor>> sztereotípiával rendelkeznek. Azonban a fent látható jelölés a jobb, hiszen átláthatóbb. Az aktorok neve a jelölésük alatt olvasható. Az alkalmazás tekintetében az aktorok szerepeket jelölnek, azonban egy személy több szerepet is betölthet, például egy alkalmazott lehet üzletvezető, de mellette, mint eladó is tevékenykedhet. Ennek fordítottja is igaz, hiszen több személy is lehet ugyanabban a szerepben. Például az AutoWorld esetében több személy is lehet ügyfél, de valószínűleg több eladó is tartozik majd a rendszerhez egyszerre. Az Ügyfél, Eladó és Üzletvezető hármas a Felhasználóval általánosítás-pontosítás viszonyban áll, amelyet egy háromszögben végződő nyíl jelöl. A nyíl az általánosabb felé mutat.

Az actorok a rendszerhez funkciókon keresztül kapcsolódnak. Ezeket a funkciókat nevezzük használati eseteknek. A használati eseteket az UML ellipszissel jelöli, melynek közepébe van írva a betöltött funkció neve. A név sokszor a funkció rövid leírása. A 29. ábrán három használati eset látható az AutoWorld vonatkozásában. A használati esetek között is fennállhat általánosítás-pontosítás viszony. Ez megjelenik a 29. ábrán is, ahol az alsó két használati eset pontosítása a fentebb lévőnek. Az Üzletvezető egyik lehetséges tevékenysége lehet az „új autó rögzítése”. Azonban ezt kétféleképp teheti meg. Erre vonatkozik a pontosítás jelen esetben, hiszen az „új autó rögzítése” esetén két lehetősége van: az „új eladó autó rögzítése” vagy „új kölcsönözhető autó rögzítése”.



29. ábra

A 30. ábrán az AutoWorld használati eset diagramja látható. Megjelenik rajta a már említett négy aktor. A diagram tartalmazza az összes használati esetet. Látható, hogy a használati esetek egy téglalapban lettek megadva, amely az alkalmazást szimbolizálja és neve is az AutoWorld lett. Egy aktor és egy használati eset között kommunikációs kapcsolat állhat fenn. Az aktorok és a használati esetek között húzott vonalak jelölik, hogy mely aktorhoz melyik használati eset tartozik. A diagramon két féle kommunikációs vonal is látható. Például az Ügyfél és az „autók listázása” között húzott vonal jelzi, hogy közöttük „oda-vissza” irányú információtovábbítás áll fenn. Azonban például az Ügyfél és az „autó lefoglalása vásárláshoz” között nyíl jelöli a kapcsolatot, amely jelzi, hogy az információ az ügyfél felől érkezik a használati eset felé. A diagram tartalmaz még egy kapcsolatot, ami szaggatott vonalú nyíllal van jelölve és <<include>> sztereotípiával rendelkezik. A kapcsolat a részfunkciókat jelölik. A nyíl a részek felől mutat az összetettebb funkció felé. Ez alapján tehát egy Eladáshoz szükség van egy Ügyfél által elvégzett foglalásra, amelyet az Eladó az eladás véglegesítésével rögzít.



30. ábra: Az AutoWorld használati eset diagramja

5.2 Állapot-átmenet diagram

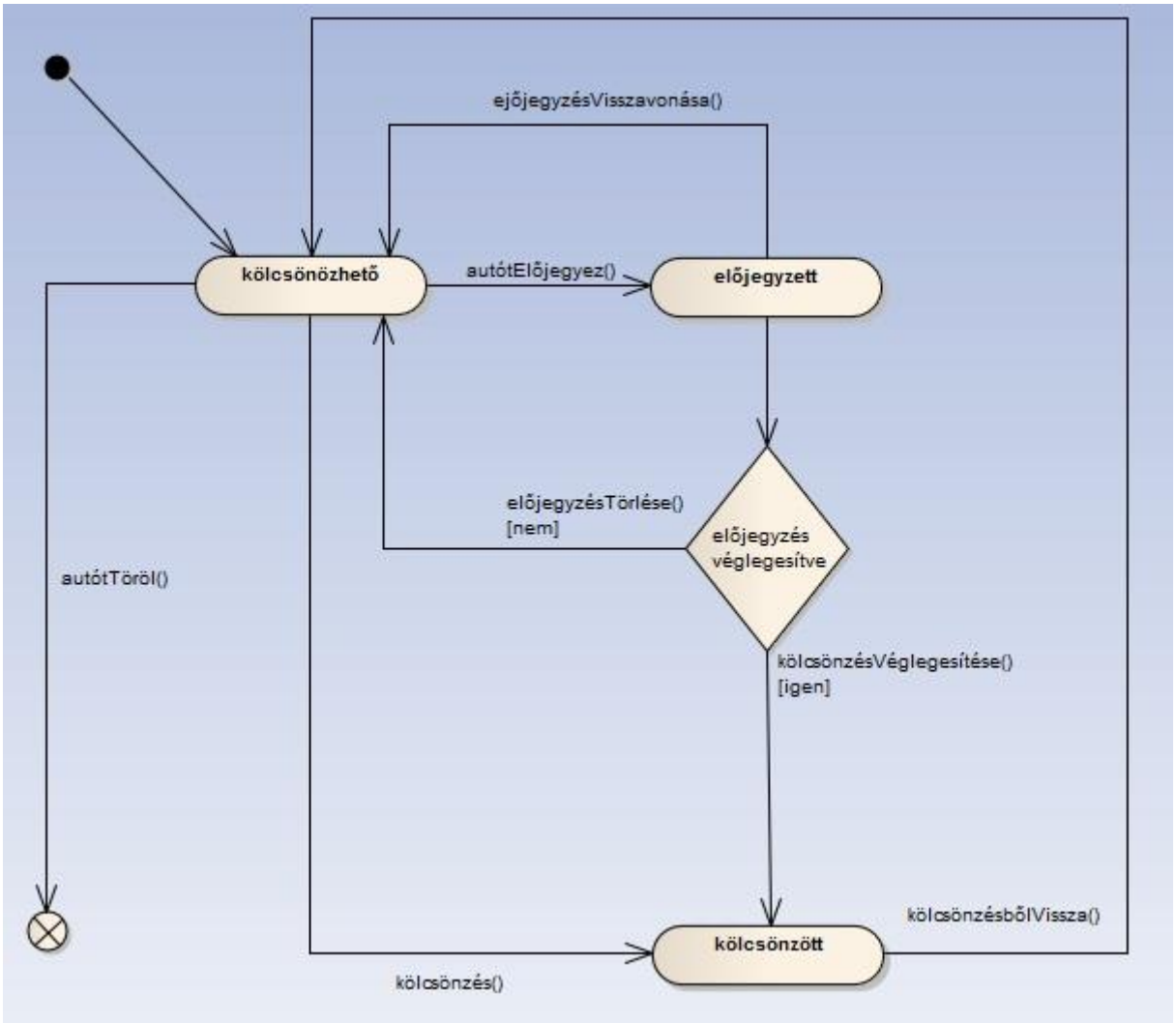
Futás közben egy alkalmazás állapottal rendelkezik, amely időben változik. Azt mondhatjuk, hogy egy alkalmazás állapota változik, ha az alkalmazás objektumainak állapota megváltozik. Ezeket az időbeli változásokat az állapot-átmenet diagram segítségével szemléletesen és egyértelműen ábrázolhatjuk.

Az objektumok állapotainak változását legegyszerűbben úgy követhetjük nyomon, ha az objektum teljes életciklusát ábrázoljuk, annak példányosításától megszűnéséig. Az objektum életciklusát tekintve számba kell venni az összes lehetséges állapotot, amelybe az objektum kerülhet. Példaként vegyük az AutoWorld rendszer „kölcsönAutó” osztályának egy objektumát, és vizsgáljuk ennek életciklusát. Eszerint a kölcsönzői autó a „kölcsönözhető”, „előjegyzett” és „kölcsönzött” állapot valamelyikébe kerülhet az életciklusa folyamán. Az állapotok jelölésének módját a 31. ábra mutatja be.

31. ábra

A 32. ábra a kölcsönautóhoz tartozó teljes állapot-átmenet diagramot szemlélteti. Ha egy „kölcsönAutó” osztályt példányosítunk, létrejön annak egy objektuma, amely kezdő állapotba kerül. A kezdőállapot az életciklust tekintve egy kitűntetett állapot, csakúgy, mint a végállapot. A kezdőállapotot a diagramon a bal felső sarokban látható fekete pont jelzi. Az ábra jól mutatja, hogy a létrejötte után a kölcsönautó azonnal „kölcsönözhető” állapotba kerül. Ezt a kezdőállapotból az említett állapotba vezető nyitott hegyű nyíl jelzi. Az adott állapotból egy másik állapotba történő állapotváltást átmenetnek nevezzük. A kezdőállapot és a „kölcsönözhető” állapot közötti átmenet például egy kiváltó ok nélkül teljesülő átmenet. Az átmenetet jelölő nyílon megjelenhet feliratként a kiváltó ok, az átmenet feltétele vagy hatása is, azonban ez opcionális. A 32. ábrán látható összes átmenetet jelző nyíl esetében meg lett

adva a kiváltó ok, amely jelen esetben egy művelet meghívása. Amennyiben az ügyfél előjegyez egy kölcsönözhető autót, akkor „előjegyzett” állapotba kerül.



32. ábra

Az „előjegyzett" állapothoz kétféle átmenet tartozik. Az előjegyzést az ügyfél visszavonhatja, míg ellenkező esetben egy választási helyzet áll elő. Ezt a döntési szituációt egy fehér rombusszal jelöli az UML. Esetünkben a kérdés az, hogy az eladó véglegesíti-e az előjegyzést. Az eldöntendő kérdést beírhatjuk a rombuszba is, de megjegyzésként is hozzárendelhetjük. A rombuszból kiinduló átmeneteken szögletes zárójelben külön meg lett adva, hogy a kérdésre adott válasz függvényében melyik átmenet hajtódik végre. Igen esetén a „kölcsönzött” állapotba kerül az objektum. Azonban jól látható, hogy közvetlenül a

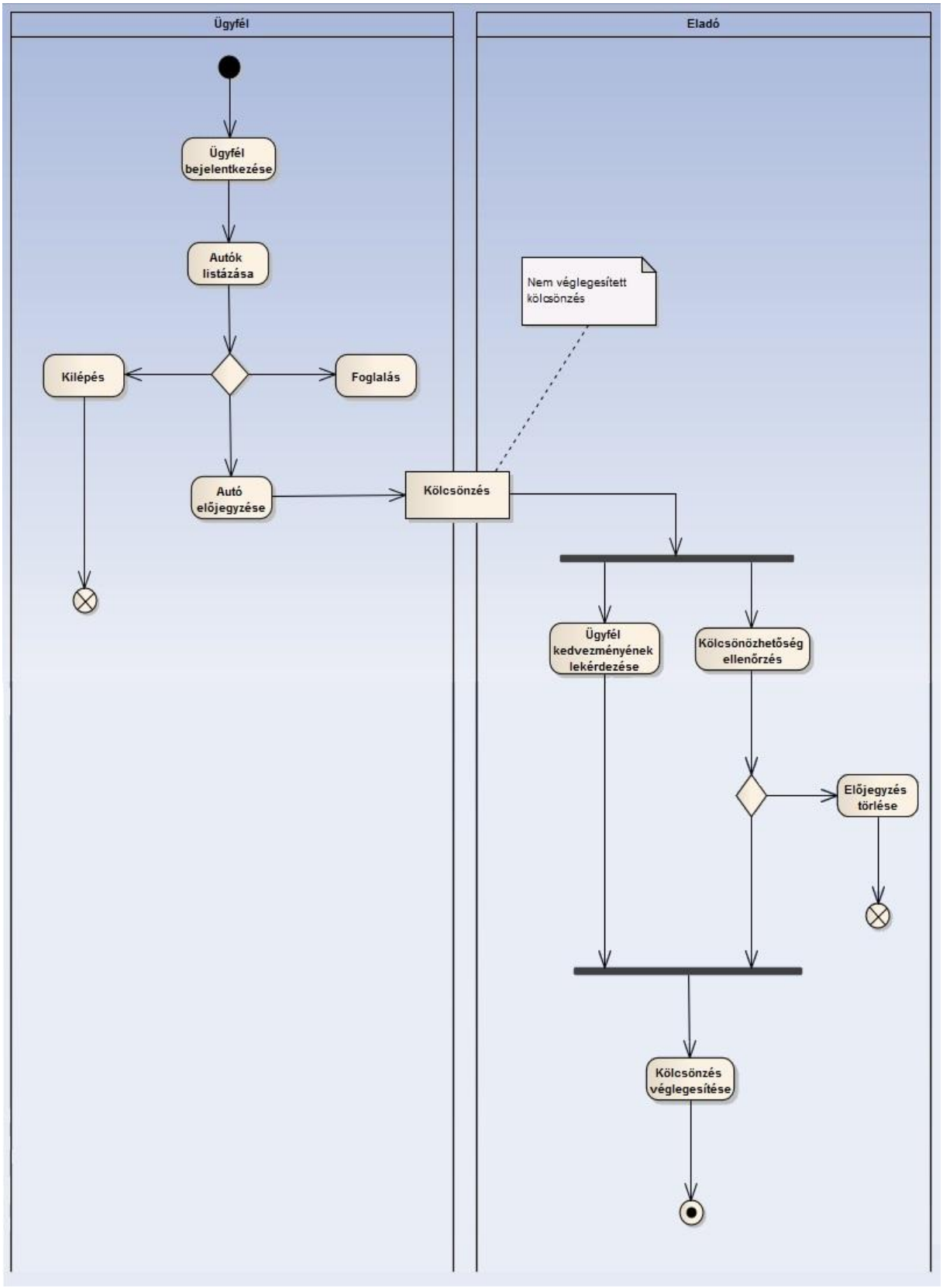
„kölcsönözhető” állapotból is eljuthattunk volna ebbe az állapotba, így ez jó példát szolgáltat arra, hogy szemléltetve legyen az, hogy egy állapot többféleképpen is bekövetkezhet.
Megfigyelhető, hogy az objektum életciklusát tekintve igen fontos állapota a „kölcsönözhető” állapot. Ugyanis az objektum létrejötte után rögtön ebbe az állapotba kerül és több-kevesebb átmenet után szintén ebbe az állapotba kerül. A rendszerből azonban törölhetőek azok az autók, amelyeket már nem szeretne az autókerekedés kölcsönbe adni. A kölcsönözhető állapotból az „autótTöröl()” átmenet a végállapotba vezet. Az UML a végállapotot a diagram bal alsó sarkában látható, bekarikázott X-szel jelöli. Kölcsön autónk esetében a végállapot egyben az objektum megszűnését is jelenti. Ha az objektum nem törlődne, a végállapotot egy fehér körben található fekete ponttal lehetne jelölni.

Az állapot-átmenet diagram az időbeli változásokat szemlélteti. Mégsem esett szó a közben eltelt időről. Az átmenet egy időpillanat alatt megy végbe, tehát nincs időtartama. Az átmenetek mindemellett félbeszakíthatatlanok, tehát atomi egységet képeznek. Az állapot viszont egy időtartamot is jelöl, amelyben az objektum nem változik.

5.3 Tevékenységdiagram

Egy tevékenységdiagram segítségével az időben lezajló változásokat, azaz a rendszer működése közbeni folyamatokat lehet leírni. A különféle folyamatokhoz megadhatjuk, hogy milyen tevékenységek alkotják. Ezen felül jól ábrázolható az is, hogy a már említett tevékenységek milyen sorrendben követik egymást, melyek futhatnak párhuzamosan, valamint mindemellett a szinkronizáció is jelölhető. Az előző részben tárgyalt állapot-átmenet diagram és a tevékenységdiagram között egy igen lényeges kiegészítő kapcsolat áll fenn. Mindkét diagramtípus az időbeli változásokat hivatott ábrázolni, azonban más módon. Az állapot-átmenet diagram esetében az állapotok állnak a figyelem középpontjában, míg az átmenetekről, azaz a tevékenységekről kevés szó esik. A tevékenységdiagram viszont éppen ennek ellenkezője. Az állapotok háttérbe szorulnak, és a köztük lévő átmenetek kapnak nagyobb hangsúlyt. A két diagramtípus közötti kapcsolat miatt nagyon sok elem van, amelyik megjelenik az állapot-átmenet diagramon is, de a tevékenységdiagramnak is fontos eszköze.

Az AutoWorld rendszer esetében vizsgáljuk meg az ügyfél által kezdeményezett kölcsönzést, mint folyamatot; és ábrázoljuk a lefutását tevékenységdiagram segítségével.


Ügyfél
Eladó
Ügyfél bejelentkezése
Autók listázása
Kilépés
Foglalás
Autó előjegyzése
Kölcsönzés
Nem véglegesített kölcsönzés
Ügyfél kedvezményének lekérdezése
Kölcsönözhetőség ellenőrzés
Előjegyzés törlése
Kölcsönzés véglegesítése

33. ábra

A 33. ábra tehát az ügyfél egy kölcsönzéséhez tartozó tevékenységdiagramot tartalmazza. Egy autó kölcsönzésében az Ügyfél és az Eladó vesz részt közvetlenül. Ennek megfelelően megjelennek a diagramon is, hiszen alapvetően az általuk elvégzett tevékenységek összessége adja a kölcsönzés folyamatát. Az Ügyfél és az Eladó egy-egy partíción belül vannak megadva, amelyek itt függőlegesen vannak ábrázolva. A partíciók segítségével különválasztható, hogy az egyes tevékenységeket melyik felhasználó végzi. A diagramon hasonlóan, mint az állapot-átmenet diagramon, a kezdő állapotot egy fekete kör jelöli. Gyakorlatilag a kezdő állapot azt jelenti, az ügyfél meglátogatja az alkalmazáshoz tartozó weboldalt. A tevékenységeket lekerekített négyszögek jelzik, ilyen négyszög jelöli például az „Ügyfél bejelentkezése" tevékenységet. A tevékenységek nyitott hegyű nyíllal vannak összekötve. A nyilak mindig a kiinduló aktivitás utáni tevékenységre mutatnak, amellyel szekvenciális végrehajtási sorrendet is jelölnek. Az ügyfél a bejelentkezés után kilistázhatja az éppen rendelkezésre álló autókat, azonban ezután válaszúthoz érkezik.

Döntenie kell, hogy foglal, kölcsönözéshez előjegyez, vagy inkább kilép. A diagramon a döntési helyzetet egy fehér rombusz jelzi, amelyből az említett három irányban léphet tovább az ügyfél, aszerint, hogy mit szeretne. Kilépés esetén ismét egy folyamvég pontban végződik a folyamat, amely az állapot-átmenet diagramon is jelen volt már. A Foglalás tevékenység még tovább részletezhető lenne, azonban most nem az a cél. Az autó előjegyzése után létrejön a rendszerben egy Kölcsönzés objektum, amely a diagramon a két partíció között egy téglalappal lett jelölve. A Kölcsönzéshez egy megjegyzés is tartozik, amely a Kölcsönzés egy pillanatnyi, elhanyagolhatatlan tulajdonságára hívja fel a figyelmet.

Ekkor kapcsolódik be az eladó a kölcsönzés folyamatába. A vezérlés átkerül az Eladó partíciójába, és ott folytatódik a folyamat. Az eladónak ekkor több feladata is van egyszerre. Az ezekhez tartozó folyamatok párhuzamosan és végrehajthatóak, amelyet a diagramon látható szinkronizációs vonal jelez. A szinkronizációs vonalat egy megvastagított vízszintes vonal jelöli. Ennek segítségével a vezérlés több szálra bontható. Itt fontos kiemelni, hogy nem kötelezően kell az „Ügyfél kedvezményének lekérdezése", és a „Kölcsönözhetőség ellenőrzés" tevékenységeknek párhuzamosan futnia. A jelölés célja csak annyi, hogy hangsúlyozzuk ezek logikailag független tevékenységek, amelyeknél lehetőség van párhuzamosságra. Az ellenőrzés során felmerül annak lehetősége, hogy a kölcsönzés mégsem valósulhat meg. Ekkor az eladó törli az előjegyzést és a folyamat véget ér, amelyet ismét egy folyamvég pont jelöl. A két szál hibátlan működését ellenőrizhetjük szinkronizációs pont

segítségével, amelyet szintén szinkronizációs vonallal lehet jelölni. A szinkronizáció sikere azt jelenti, hogy a kölcsönzés megfelelő. A szálak egyesülnek, és a vezérlés átkerül az eladó véglegesítési tevékenységére. Ezzel a kölcsönzés folyamata nemcsak sikeres lesz, hanem be is fejeződik, amelyet egy az állapot-átmenet diagramnál is megismert végállapot jelöl. Egy végállapot elérése során befejeződik a folyamat, azonban ezzel egyidejűleg megszakadnak az esetlegesen még futó szálak is.

5.4 Interakció diagramok

Egy interakció diagram üzenetváltásokat ábrázol. Az üzenetváltások több egymással kapcsolatban lévő partner között mehetnek végbe. A partner lehet objektum, aktor, osztály és még sok más elem. Interakció diagrammal leírhatóak követelmények és tesztesetek, valamint remekül lehet megjeleníteni és vizsgálni a rendszer egyes folyamatait. Az UML három meghatározó interakció diagramja a szekvenciadiagram, a kommunikációs és az idődiagram. Az említett három diagramtípus mindegyike használható ugyanahhoz a folyamathoz tartozó üzenetváltások leírására, azonban mindhárom különböző nézőpontból közelít.

- Szekvenciadiagram:

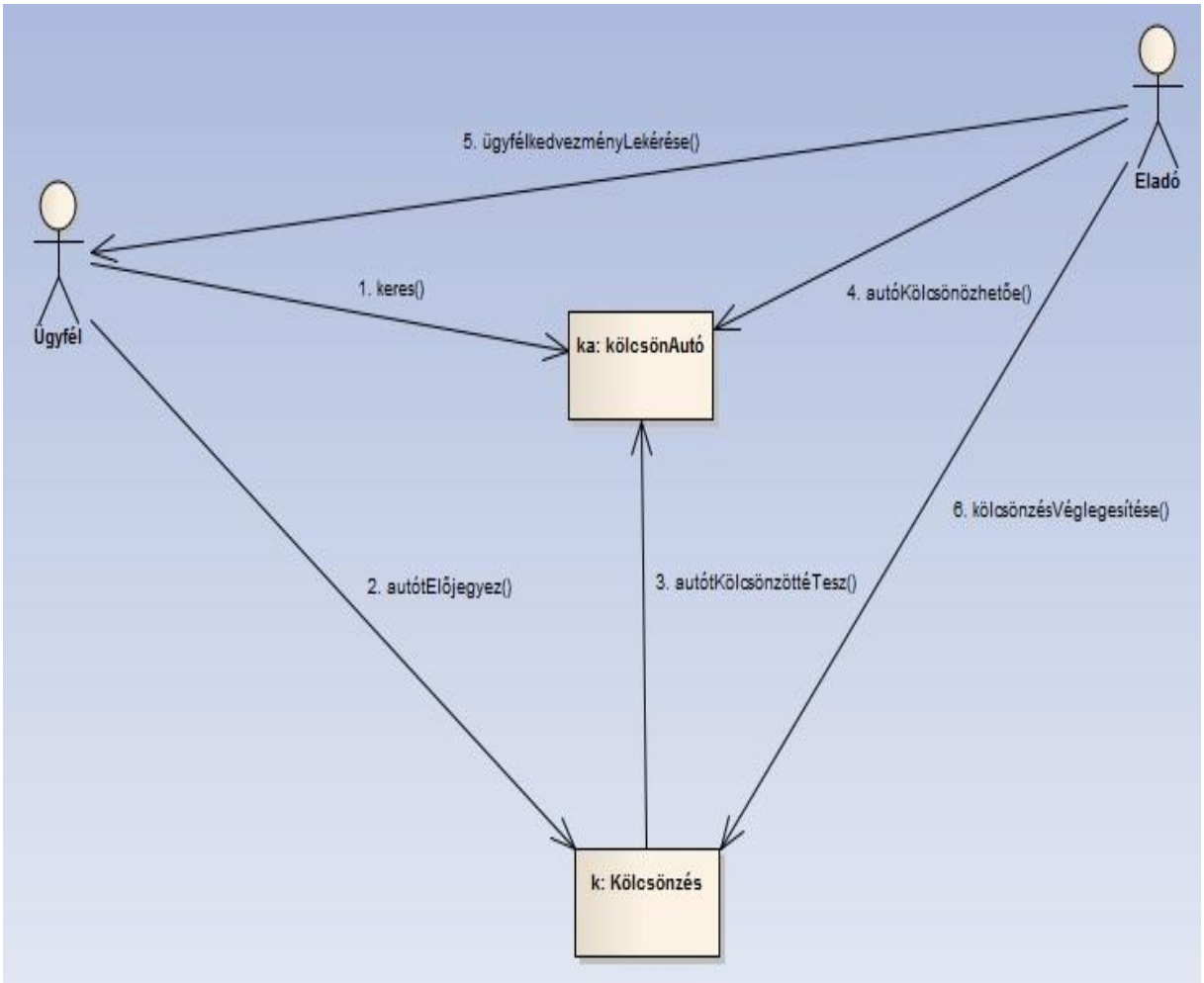
  Szekvenciadiagramok remekül használhatóak, ha az interakció viszonylag kevés partner között zajlik, és ezen partnerek között sok üzenetváltás megy végbe. Azonban az idő és a partnerek állapotai nehezen ábrázolhatóak a segítségével.

- Kommunikációs diagram:

  Kommunikációs diagramok használata ajánlott, ha egyszerű interakciókat szeretnénk modellezni interakciós partnerek kiterjedt vagy komplex hálózatán belül. Ennek a diagramtípusnak a hátránya, hogy ha az ábrázolandó üzenetek száma meghalad egy bizonyos mennyiséget, a diagram veszít átláthatóságából.

- Idődiagram:

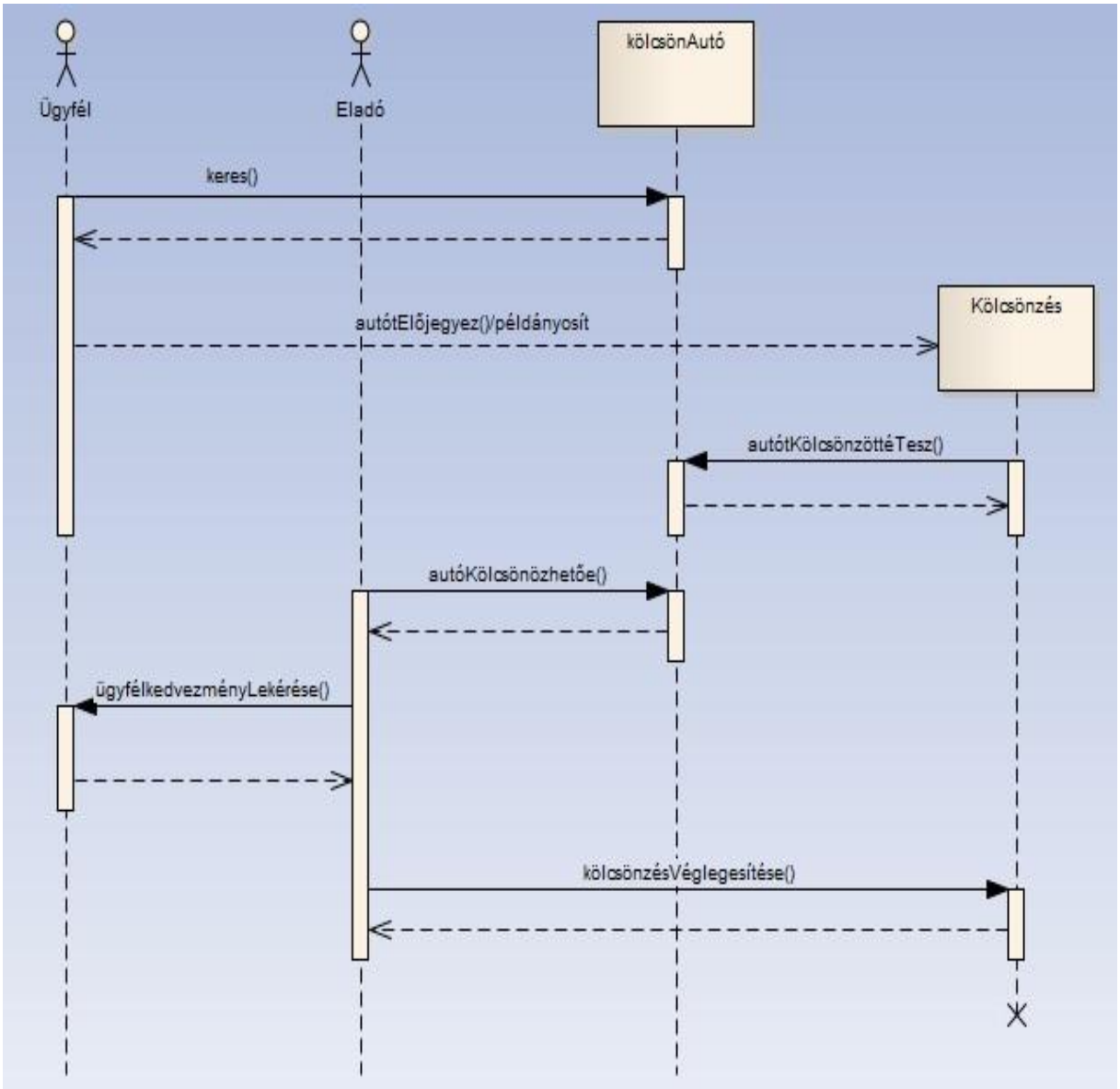
  Idődiagram használata akkor indokolt, ha kevés partner közötti üzenetváltást szeretnénk szemléltetni, valamint a különböző üzenetváltások időbeni megjelenését szeretnénk hangsúlyozni. A partnerek állapotai megjelenhetnek, amellyel még szemléletesebben írhatóak le a folyamatok, hiszen ábrázolható, hogy melyik partner melyik időpillanatban milyen üzenetet küldött.



34. ábra

A 34. ábra az AutoWorld rendszer kölcsönzési folyamatához tartozó kommunikációs diagramot ábrázolja. A kölcsönzésben egy ügyfél és egy eladó vesz részt, amelyek mint aktorok jelennek meg a diagramon. A kölcsönzés folyamatához tartozik még maga egy példányosított Kölcsönzés, amely a diagramon egy ’k’ nevű objektumként van jelölve. A

kölcsönzés során az ügyfél a ’ka’ nevű kölcsönAutó példányt szeretné kölcsönözni. A kölcsönzésben résztvevő négy partner közötti üzenetváltások nyíllal vannak jelölve. A nyilakon található címkék jelzik, hogy milyenek a küldött üzenetek. Jelen esetben ezen üzenetek metódushívásokat foglalnak magukban. A nyilak címkéi előtt szereplő számok egy kezdetleges időbeli sorrendet szemléltetnek, természetesen a nagyobb sorszámmal rendelkező üzenet időben később kerül továbbításra.



35. ábra

A 35. ábra a tervezett rendszer kölcsönzési folyamatát mutatja be szekvenciadiagram segítségével. Hasonlóan a kommunikációs diagramhoz ezen a diagramon is aktorként tűnik fel a kölcsönzésben résztvevő ügyfél és eladó, továbbá a kölcsönAutó és a kölcsönzés, mint a rendszer egy-egy objektuma. A négy résztvevőhöz kapcsolódik egy életvonal, amely az időrendiséget szemlélteti. Tehát az életvonalon történő lefelé való haladást az idő múlásának lehet tekinteni. Az életvonalat az interakciós partnert jelző ikonból indított, lefelé haladó szaggatott vonallal jelezzük. Az életvonalak mentén esemény-előfordulások találhatóak. Ezen események lehetnek üzenetek küldései vagy fogadásai. Két interakciós partner közötti üzenetváltást egy nyíl jelöl. A nyílon szereplő név az üzenet neve, amely jelen esetben metódushívás. Szaggatott vonalú nyíl jelzi az üzenetet kapó partner által küldött választ. Szaggatott vonalú nyíl mutat az Ügyfél felől a Kölcsönzés felé. Azonban ez mégsem válasz, hanem egy példányosítási üzenet, melynek következtében létrejön az objektum. A diagramon szereplő nyilak mind metódushívások és az azokra érkező válaszok. A „kölcsönzésVéglegesítése()” metódus után a Kölcsönzés véglegesítődik, ezáltal tárolódni fog az adatbázisban. Ennek következtében valamikor a válasz visszaküldése után – egy meghatározatlan időpontban – megszűnik majd a Kölcsönzés objektuma, amelyet az életvonalának végén elhelyezett nagy X jelez.

6. Összefoglalás

Dolgozatom célja az volt, hogy bemutassam az UML által nyújtott eszközöket. Munkám során egy általam tervezett rendszert használtam példaként, amely reményeim szerint megfelelő volt a példák megértése során. Elsődleges célom a leggyakrabban használt diagramok bemutatása volt, amelyet szerintem sikerült teljesítenem. Ugyan maradtak meg nem említett diagramok, de véleményem szerint az említett diagramok elegendőek egy kezdő fejlesztő számára. Az egyes diagramok esetében előfordulhat, hogy nem mutattam be az összes eszközt, mivel a megvalósítandó rendszer vonatkozásában nem éreztem szükségesnek, de ahol volt értelme röviden igyekeztem írni a példán túli eszközökről is.

Végül szeretném megköszönni Pánovics Jánosnak, a szakdolgozatom témavezetőjeként és az egyetemi éveim alatt tanáromként nyújtott segítségét és türelmét irántam.

7. Irodalomjegyzék

1. Harald Störrle: UML2 für Studenten, Pearson Studium, 2005,
   magyar fordítása Adamkó Attila - Bicskey Simon - Espák Miklós: UML2, Panem, 2007

2. Vég Csaba: Alkalmazásfejlesztés, Logos 2000 Bt., 1999

3. Raffai Mária: UML2 Modellező nyelvi kézikönyv, Palatia Nyomda és kiadó, 2007

4. Sike Sándor-Varga László: Szoftvertechnológia és UML, ELTE Eötvös Kiadó, 2001

5. http://www.rational.hu/uml

6. http://www.uml.com