SZAKDOLGOZAT

KOVÁCS BALÁZS

DEBRECEN
2008

DEBRECENI EGYETEM
INFORMATIKAI KAR

Az UML eszközeinek bemutatása a NetCafé kávéházi rendszer tervezésén keresztül

Témavezető:
Pánovics János
egyetemi tanársegéd

Készítette:
Kovács Balázs
programtervező informatikus

Debrecen, 2008

Tartalomjegyzék

1 BEVEZETÉS

1.1 Mi az UML?

A mai világban sok informatikus az UML-t és az MDA-t (Model Driven Architecture) a szoftvergyártás jövőjének tekinti. Ezeknek az eszközöknek a segítségével óriási lépést tehetünk az egyes szoftverfázisokhoz kötődő tevékenységek automatizálásához. Tulajdonképpen az alkalmazástervezés és fejlesztés folyamatainak bizonyos fokú egyszerűsödését, könnyebbé tételét szolgálják ezek a tervezői eszközök. Az UML egy objektumorientáltságra épülő nyelv, mely egy elemző és tervező eszköz. Az UML egy grafikus modellező eszköz, azaz a tervezett rendszer, alkalmazás modell szintű megjelenését diagramok segítségével valósítja meg, segítségével csupán az alkalmazás vázlatát készíthetjük el, a teljes implementációt nem. Az UML széles körű, integrált eszközöket nyújt, amelyek segítségével a szoftver bizonyos fokú definiáltsága érhető el. Az UML olyan eszközrendszert nyújt, amelynek segítségével

- az alkalmazásfejlesztők specifikálhatják, érvényre juttathatják a kifejlesztendő rendszerrel kapcsolatos követelményeket, elvárásokat
- a rendszer különböző szemléletű és megközelítésű modelljeit konstruálhatják meg (architektúra, viselkedés, szolgáltatás)
- szoftver-életciklus szakaszaihoz szükséges dokumentációk elkészíthetők

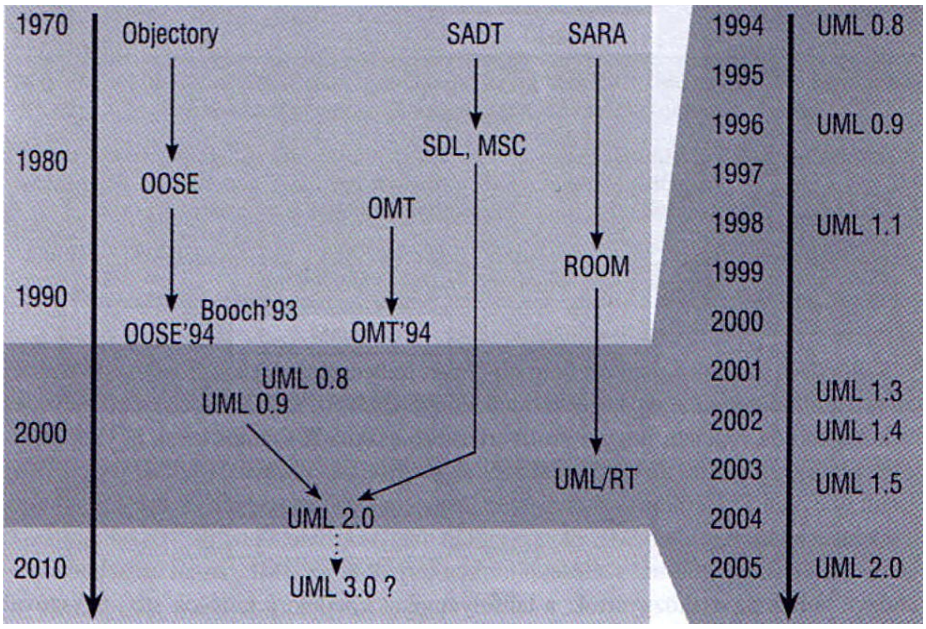
1.2 Az UML története, hozadéka

A 70-es évek elején jelentek meg az első szoftverfejlesztési módszertanok (strukturált programozás), majd a 80-as évektől pedig az első objektumorientált módszerek. Az objektumorientáltság szemlélet lényege, hogy a valós világot modellezzük. A valós világban egyedek élnek, ezeket az egyedeket osztályozzuk, csoportba foglaljuk valamilyen közös

tulajdonság alapján. Mivel a valós világ nagyon összetett, nem tudjuk teljes egészében megérteni, átlátni, ezért a mi szempontunkból lényegtelen tulajdonságokat elhanyagoljuk. A 90-es évek elejére már tucatnyi különböző módszertan létezett. Majd lassacskán megindult az UML története:

- 1994: Grady Booch és Jim Rumbaugh (OMT) a nyelvek egyesítésének céljából megkezdik az egységes modellezőnyelv kidolgozását, melynek neve Unified Method
- nem sokkal később csatlakozik Ivar Jacobson (OOSE) és 1995-ben a szabványosítási törekvéseket az OMG szerezte meg
- OMG kiadta az UML 0.9 verzióját
- valamint megalakul a legnagyobb informatikai cégek (Microsoft, Oracle, HP, DEC, Rational Software, stb.) támogatásával az UML Partners konzorcium, az UML kidolgozásának támogatásának céljából
- 1997: elkészül az UML 1.0-ás verziójának specifikációja, majd az 1.1-es verzió, melyeket az OMT szabványosít
- 1998: elkészül a 1.2-es verzió
- 1999: elkészül az 1.3-as verzió
- 2000: elkészül az 1.4-es verzió
- 2000/2001: elkezdődik az UML 2.0-ás verziójának kidolgozása
- 2003: elkészül az 1.5-ös verzió
- 2004: az UML 2.0 dokumentációjának elkészülése
- 2005: az UML 2.0 megjelenése

Az UML hozadéka a különböző tudásterületek integrálása, és ezek megszilárdítása. Hiszen az UML fogalmai közül már sok létezett pl: szekvenciadiagram az ITU-T szabvány az MSC-kből formálódott, osztálydiagramok az ER-diagramok utódjai stb. A különböző megközelítések integrálása valósult meg az UML modellező nyelvben.



(1.2. ábra)

1.3 Az UML felépítése

Az UML felépítése egy metamodellezési megközelítésen alapszik. A metamodellezés azt jelenti, hogy minden konkrét rendszer egy modell példánya, minden modell egy metamodell példánya, és tovább folytatva minden metamodell egy meta-metamodell példánya (MOF Meta Object Facility). A metamodellezési konstrukció segítségével így a nyelv tömör maradhat.
Az UML különböző diagramjai a modellezett rendszert különböző aspektusokból szemléltetik. Így előfordulhat, hogy az egyes diagramok között átfedések lehetnek, azaz megtörténhet, hogy a rendszer ugyanazon pontját modellezzük rajtuk, de a diagramok típusától függően ugyanazt a dolgot más megvilágításban láthatjuk.

Alapvetően az UML két nagy csoportba osztja a diagramokat:

- Struktúrára vonatkozó diagramok: A strukturális diagramok a rendszer statikus jellemzőit, a statikus elemeket és ezek kapcsolatait, ezáltal a rendszer belső struktúráját modellezik: a legfontosabb diagramtípus az osztálydiagram. Minden más struktúradiagram az osztálydiagram egy speciális változatának tekinthető (objektumdiagram, architektúradiagramok)

- Viselkedésre vonatkozó diagramok: A viselkedési diagramokkal a strukturális diagramokon modellezett elemek dinamikáját, a rendszer működése közben megvalósított viselkedését modellezhetjük. Ezek a diagramok írják le, hogy az erőforrások hogyan hajtják végre a feladataikat, hogyan működnek együtt a rendszer céljainak megvalósítása közben: használati eset diagram, állapotautomata diagram, tevékenység diagram (aktivitás-diagram), interakció diagramok (szekvencia, kommunikációs, időzítési)

Természetesen lehetetlenség az összes, az alkalmazásfejlesztés során felmerülő fogalomra pontos kifejezési formát, jelölést, vagy eszközt találni. Ennek ellensúlyozására az UML megengedi, hogy bármely modellelemhez kommentet, tájékoztató jellegű szöveges magyarázatot fűzzünk. A kommenteket egy szaggatott vonallal a modellelemhez kötött téglalapban helyezhetjük el.

1.4 Az UML környezete

A szoftver előállításának folyamatát fejlesztési vagy szoftverfolyamatnak nevezzük. Tehát a szoftvernek is van életciklusa. Az életciklus szakaszokra bomlik, ezeket fázisoknak nevezzük. Az UML olyan széleskörű és integrált eszközrendszert nyújt, hogy az életciklus bármely szakaszában ezen eszközök alkalmazhatóak:

- Projektdefiníció: lehet egy teljesen új rendszer definiálása, vagy meglévő rendszer újrafogalmazása, módosításának igénye
- Elemzés: az elemzési fázison derül ki, hogy voltaképp mi is a feladat. Mi a célja a megtervezendő rendszernek, ehhez milyen információkra van szükség: interjúk, követelmények feltárása, folyamatok leírása stb.
- Tervezés: az elemzés és tervezés határai nem válnak el egymástól. Itt már a technikai döntések játszanak fontos szerepet, azaz, hogy hogyan valósítsuk meg.
- Megvalósítás: erőteljesen a programozás határozza meg, vagyis a konkrét technológia kiválasztása és megvalósítása

- Integráció: az újonnan kifejlesztett rendszer már meglévő rendszerekhez való illesztése
- Bevezetés: az alkalmazás, használat megkezdése. Különböző tesztek alkalmazása a pl: funkcionalitás ellenőrzésére
- Karbantartás: az alkalmazás működése közben felszínre került hibák, esetleg felmerülő új követelmények, amelyeket kezelni kell
- Újratervezés és leállítás: egy szoftver újbóli tervezése vagy a szoftver használatának megszüntetése

1.5 Kiterjesztési mechanizmusok

Az UML szabványos jelölései a modellek ábrázolásának mindössze egy általános keretét határozzák meg. A kiterjesztési mechanizmusok teszik lehetővé, hogy az általános jelölésrendszert a fejlesztés által megkövetelt irányba specializáljuk. Ennek segítségével kiegészíthetjük a modellünket, az UML-t pedig alkalmassá tehetjük arra, hogy a szabvány keretein belül az alkalmazás szakterületének, az alkalmazott technológiának és a fejlesztési módszernek megfelelő jelölésekkel is rendelkezzen. Az UML a következő három kiterjesztési mechanizmust definiálja: sztereotípia, megszorítás, kulcsszavas értékek.

- A sztereotípia egy olyan eszköz, amelynek segítségével az alkalmazás elemei magas szinten tipizálhatóak, azaz általános céljuknak és jellegüknek megfelelően csoportosíthatóak. Olyan új építőelemek bevezetését teszi lehetővé, amelyek már meglévő elemekre épülnek, de segítségükkel az épített modell specializációja válik lehetővé. A sztereotípiákat francia idézőjelek között adjuk meg
- A megszorításokkal (OCL nyelv, Object Constraint Language) a modellelemek olyan jellemzőit adhatjuk meg, amelyeket másképpen nem tudunk kifejezni. A megszorítás olyan általános eszköz, melynek segítségével a modellünket pontosítani tudjuk, egy olyan körülményt tudunk kifejezni segítségével, amely körülménynek a modellelemre vonatkozóan, annak teljes életciklusa során fenn kell állnia. Megszorításokat a modellelemek mögött kapcsos zárójelek között adhatjuk meg

- A kulcsszavas érték egy név-érték pár, amellyel a modellek elemeit lehet megjelölni. A kulcsszavas érékeket tipikusan a kulcsszó = érték formában adhatjuk meg

1.6 A tervezett rendszer

A NetCafé egy kávéház által használt számítógépes rendszer, amely nyilvántartja a kávéház teljes raktárkészletét, felügyeli a kávék, teák, fagylaltok és sütemények elkészítését, illetve értékesítését. (Ebben a kávéházban a pincér csak kiviszi a megrendelt tételeket; a vendég az asztalon elhelyezett számítógép segítségével rendel, illetve a kávéházban tartózkodása alatt szabadon használhatja az internetet (az internet használatát a rendszer nem tartja nyilván). A rendszer használatához a felhasználóknak regisztrálni kell, ezt az üzletben dolgozó „rendszergazda" végzi el, amely során a felhasználó kap egy felhasználói nevet és jelszót. Adott felhasználó rendeléseit a rendszer rögzíti. A rendszer webes felületén keresztül a vendégeknek lehetőségük van a különböző kávé- és teakülönlegességek, illetve édességek közti válogatásra és ezek megrendelésére. A vendég kifejezheti elégedettségét: 1-től 10-ig terjedő skálán az általa kiválasztott elégedettségi szint szintén tárolásra kerül. Többszöri vásárlás esetén a hozzá tartozó elégedettségi szint frissül (átlagolódik). A NetCafé rendszernek a vendégek kiszolgálása mellett a kávéház raktárkészletét is nyilván kell tartania. Ehhez szükséges tudni, hogy mely alapanyagokból mennyi áll rendelkezésre és melyekből kell további egységeket rendelni (a receptek tükrében).
A süteményeket, illetve a fagylaltokat készen vesszük, a teákat és kávékat azonban helyben készítjük el, ezért a raktárkészletben az ezekhez szükséges alapanyagokat kell kezelni. A raktárban figyelembe kell venni az egyes termékek romlandóságát; felhasználhatósági időn túl a raktárban lévő alapanyagokat le kell selejtezni és elszállíttatni.

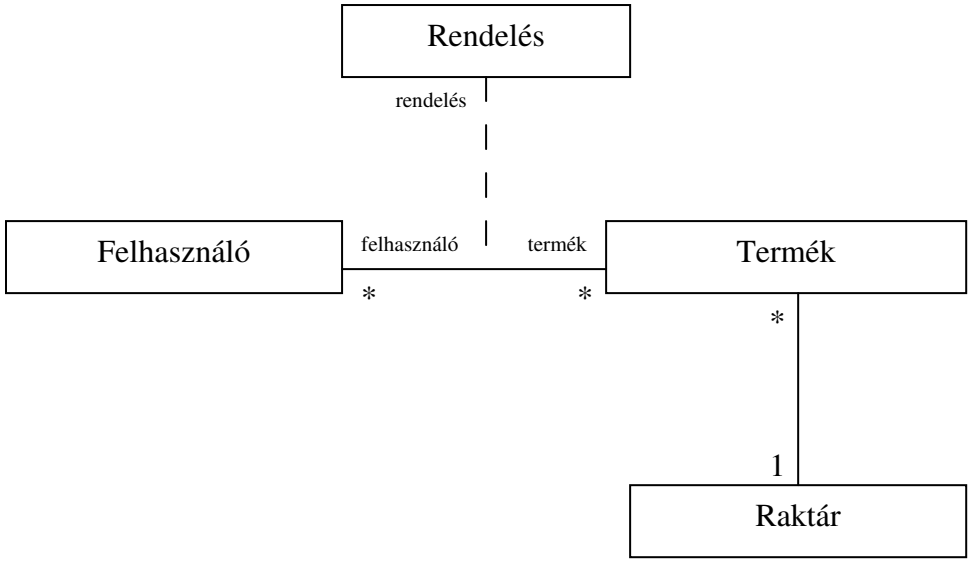
2 OSZTÁLYOK ÉS KAPCSOLATOK

Az osztálydiagramok a kifejlesztendő rendszer statikus részének, szerkezetének leírására szolgálnak. A szoftveréletciklus bármely fázisában alkalmazhatóak. Az osztály, mint fogalom az objektumorientáltság egyik fontos része. Az OO középpontjában az áll, hogy a valós világ objektumait modellezzük, vagyis a való világban előforduló szakmai „objektumokat" leképezzük technikai objektumokra. Az osztálydiagramokat a modellek gerincének tekintik.
A szoftveréletciklus fázisai alapján megkülönböztetünk különböző osztálydiagramokat:

- Elemzés: elemzési szinten az osztályok nem mások, mint a szakterület fogalmai. Lényeg a probléma megértése és dokumentálása
- Tervezés: a szakmai struktúrák technikai megvalósításáról van szó. Tehát technikai aspektusok jelennek meg az osztálydiagramokon
- Megvalósítás: konkrétan hogyan válik valóra egy megoldás. Tehát a technológia alkalmazása, vagyis az osztályok egy implementációs nyelvet jelentenek

2.1 Elemzési osztálydiagram

Az alkalmazások a világ egy szeletére, kis részére vonatkoznak, amit szakterületnek nevezünk. Egy szakterület lényeges fogalmai és kapcsolatai egyszerű osztálydiagrammal ábrázolhatók, melyben a szakmai fogalmaknak, illetve szakterületi folyamatnak egy **osztály** felel meg. Az osztály az objektumorientált világban megjelenő fogalom, amely azt jelenti, hogy a modellezés során a „leegyszerűsített" objektumokat valamilyen közös tulajdonság alapján csoportosítjuk.



(2.1.1. ábra)

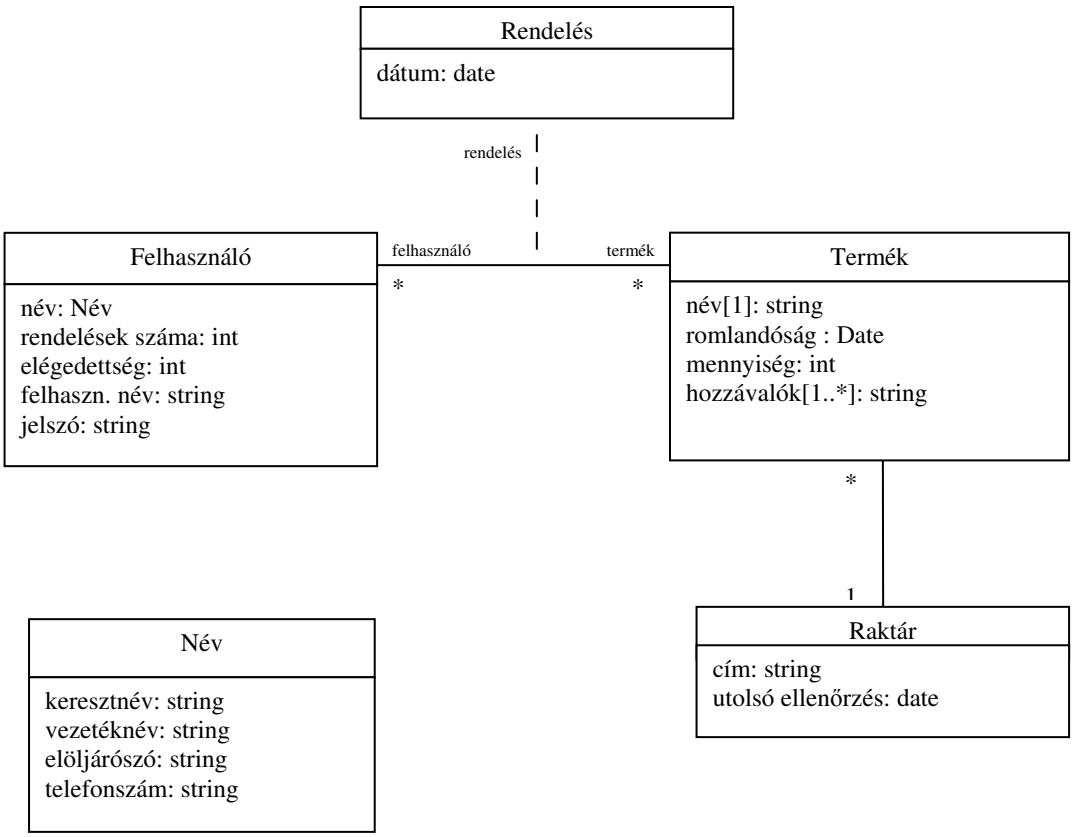
A diagram téglalapjai osztályokat ábrázolnak, amelyeket fogalmakként lehet felfogni. Tehát a 2.1.1. ábra azt fejezi ki, hogy szakterületen beszélhetünk felhasználókról, rendelésekről, termékekről, raktárról.

Két osztály közötti vonal egy **asszociáció**, amely az osztályok közötti kapcsolat kifejezésére szolgál. Az asszociációk nagy részében a résztvevő osztályok egyenrangúak, azaz egyik osztály sem alárendeltje a másiknak, és egymástól függetlenek, csak kommunikálnak, információt cserélnek egymással. A vonalak végén feliratok állhatnak, amelyek szerepköröket fejeznek ki. Szintén a vonalak végén számosságot kifejező karakterek állhatnak:

- az alapértelmezett számosság 1, ami elhagyható, a * tetszőleges mennyiséget fejez ki
- a számosság kifejezésére intervallumot is megadhatunk. Az intervallum határai tetszőleges természetes számok lehetnek, beleértve a 0-t; illetve a maximum érték nem lehet kisebb a minimum értéknél pl: 0..1

A nyíl egy olyan asszociáció, amelyen csak egy irányban navigálhatunk. A fenti ábrán minden asszociáció kétirányú, tehát mindkét irányba navigálhatunk. A 2.1.1. ábra asszociációi binárisak, de lehetnek ternáris kapcsolatok is. Ezt a fajta kapcsolatot egy rombusz segítségével tudjuk megadni. Különleges osztály a „Rendelés”, ami nem csupán osztály,

hanem egyben asszociáció a „Felhasználó” és a „Termékek között”, ezért asszociációs osztálynak nevezzük, és szaggatott vonallal jelöljük. A 2.1.1. ábra a szakterület csak fontosabb fogalmait és a köztük lévő kapcsolatokat szemlélteti.
A fogalmak pontosabb leírására **attribútumok** szolgálnak, melyek külön rekeszbe kerülnek.



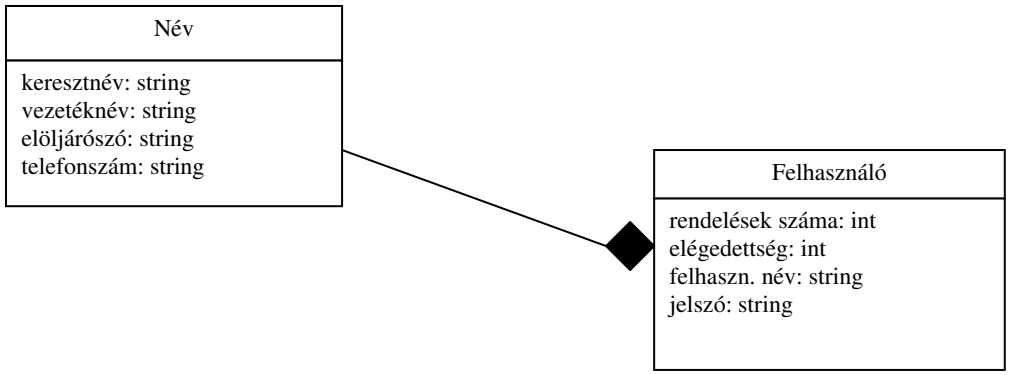
(2.1.2. ábra)

Az attribútumokat típussal is el lehet látni: lehet altípus, vagy akár más osztály is. A típus segítségével az attribútum által felvehető értékek tartományát határozzuk meg. A típus lehet az UML valamely típusa, egy programozási nyelvből származó típus, vagy a modellben szereplő objektumtípus. Ezen kívül az attribútumok számossággal is elláthatók. A multiplicitás azt fejezi ki, hogy az adott attribútumhoz hány érték tartozhat. A multiplicitást szögletes zárójelek között adhatjuk meg egy tartománnyal, azaz a tartomány alsó és felső határának meghatározásával.

Az osztályok közötti kapcsolat egy speciális fajtája a **kompozíció**. A kompozíció egy egyszerű rész-egész kapcsolat. Az asszociáció tulajdonképpen egy hivatkozást jelent, a kompozíció egy tartalmazási jelentéssel bír:

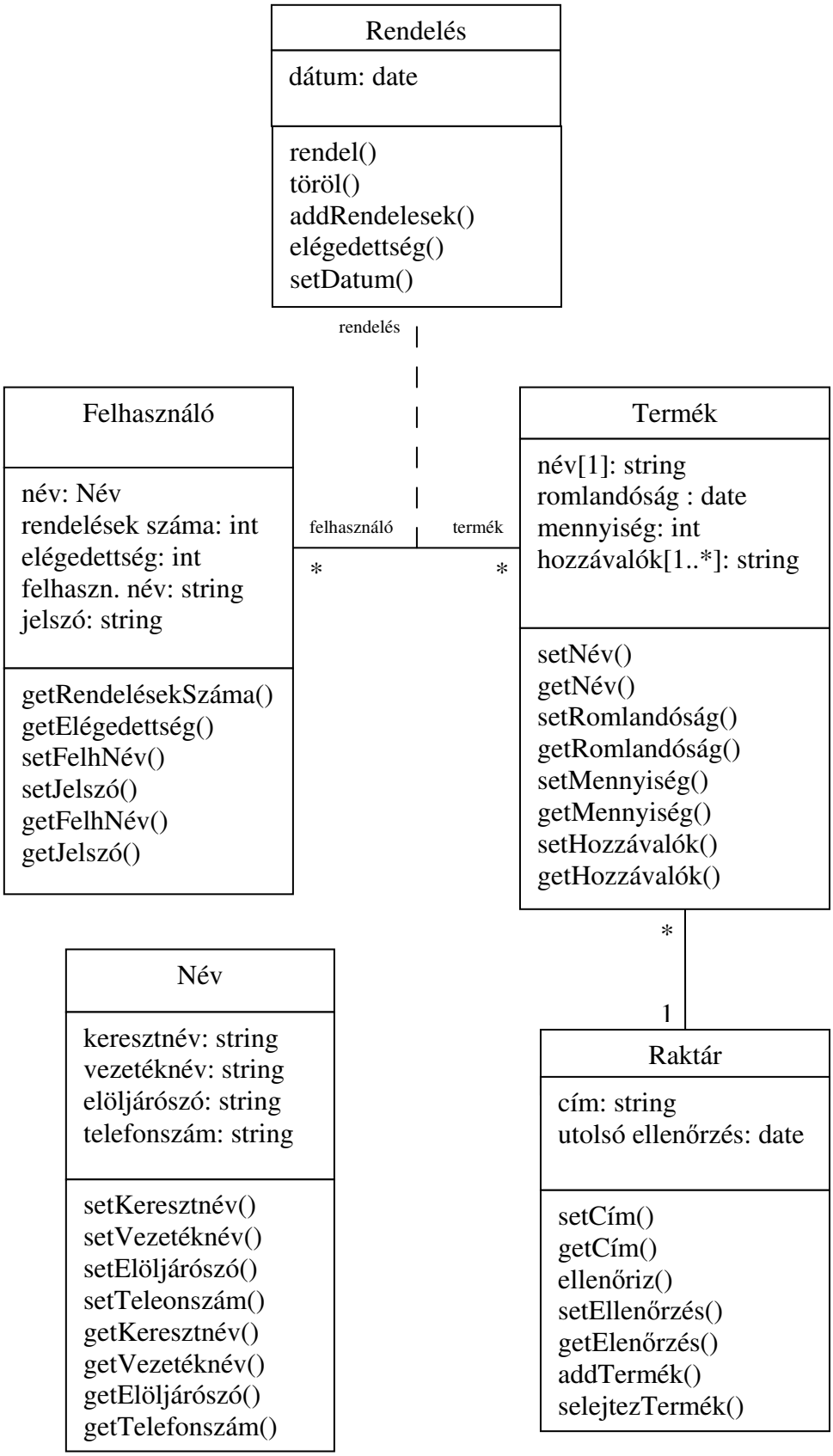
- A részobjektum teljesen B egészobjektum birtokában van. Semmilyen más objektum nem birtokolhatja A-t B-n kívül
- B törlése esetén A-nak is törlődnie kell
- egy rész csak akkor törlődhet, ha vagy törlődik B egész rész, vagy a részek számossága megengedi

A kompozíciót az asszociáció végénél elhelyezett fekete rombusszal ábrázoljuk. Másik ábrázolásmódja, ha egy attribútumot összetettként tüntetünk fel: kapcsos zárójelek között composite kulcsszóval. A kompozíció mindig bináris kapcsolat. A 2.1.3. ébrén láthatjuk, hogy a „Felhasználó” magába foglalja a „Név” fogalmát.



(2.1.3. ábra)

Az előzőekben csak a szakterület statikus megközelítését próbáltam vázolni. Az OO világban azonban egy objektumoknak nemcsak struktúrája van, hanem viselkedésmódja is. Az osztályok viselkedése **metódusokkal** írható le. A metódusok számára szintén külön rekeszt toldunk az osztályhoz az osztálydiagramban. A viselkedésmódok implementációjáról az osztálydiagram semmilyen információt nem tartalmaz, csupán az osztály objektumai által kínált szolgáltatások, viselkedések meghívásának feltételeit modellezi.


Rendelés
dátum: date
rendel()
töröl()
addRendelesek()
elégedettség()
setDatum()
rendelés
Felhasználó
név: Név
rendelések száma: int
elégedettség: int
felhaszn. név: string
jelszó: string
getRendelésekSzáma()
getElégedettség()
setFelhNév()
setJelszó()
getFelhNév()
getJelszó()
felhasználó
termék
*
*
Termék
név[1]: string
romlandóság : date
mennyiség: int
hozzávalók[1..*]: string
setNév()
getNév()
setRomlandóság()
getRomlandóság()
setMennyiség()
getMennyiség()
setHozzávalók()
getHozzávalók()
*
1
Név
keresztnév: string
vezetéknév: string
elöljáró: string
telefonszám: string
setKeresztnév()
setVezetéknév()
setElöljáró()
setTeleonszám()
getKeresztnév()
getVezetéknév()
getElöljáró()
getTelefonszám()
Raktár
cím: string
utolsó ellenőrzés: date
setCím()
getCím()
ellenőriz()
setEllenőrzés()
getElenőrzés()
addTermék()
selejtezTermék()

(2.1.4. ábra)

Az objektumorientáltság egyik központi fogalma az **öröklődés**. Az osztályok tulajdonságainak összegyűjtése során észrevehetjük, hogy bizonyos jellemzők, viselkedések több osztályban ismétlődnek. Ha találunk olyan osztályokat, amelyek közös attribútumokkal, műveletekkel és asszociációkkal rendelkeznek, akkor lehetséges, hogy az osztályok által képviselt fogalmaknak létezik közös, általános fogalma. Ekkor ezeket a közös jellemzőket kiemelhetjük egy olyan osztályba, amelyet az általános fogalomra utaló névvel látunk el, és összekapcsolhatjuk, egy speciális viszonyt létesíthetünk azokkal az osztályokkal, amelyekből a közös tulajdonságokat kiemeltük. Az általánosítás egy viszonyt fejez ki a szuperosztály és annak egy vagy több alosztálya között. Az alosztály örökli a szuperosztály attribútumait és műveleteit, és az alosztály példányai minden olyan helyen előfordulhatnak, ahol a szuperosztály egy példánya előfordulhat. Tehát UML általánosításról (Generalization) beszél. Az öröklődést fehér végű nyíl jelzi, amely a speciális felöl halad az általános felé.
Az öröklődés kapcsán megjelenik az absztrakt osztály fogalma, ami olyan osztály, amely nem példányosítható. Az absztrakt osztályt dőlt betűvel jelöljük. Az általánosítási kapcsolatok grafikusan egybefoghatók oly módon, hogy egyetlen közös nyíllal vannak ábrázolva. Ezt akkor lehet megtenni, ha ugyanazon általánosításhalmazhoz tartoznak (Generalization Set), amit a szaggatott vonal jelez. Az általánosításhalmazok különböző tulajdonságokkal rendelkezhetnek:

- disjoint: az általánosításhalmazon belüli általánosítások diszjunktak. A disjoin t ellentéte az overlapping. Alapértelmezésben disjoint van
- complete: azt fejezi ki, hogy nincsenek további alosztályok. A complete ellentéte az incomplete

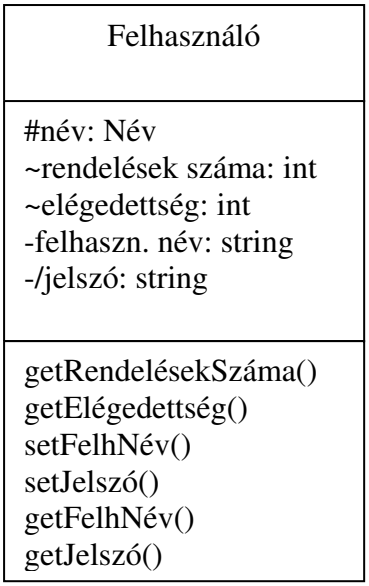
2.2 Tervezési osztálydiagram

Az elemzési fázisban arról esik szó, hogy mit kell megvalósítani, addig a tervezési fázisban ennek megvalósításáról esik szó. Az elemzési modellek a problémát írják le, a tervezési modellek pedig a megoldást. Így a leírásnak részletesebbnek kell lennie, és a megvalósítás irányába kell hogy mutasson modellezés. Az elemzési osztálydiagramok eszközei közül néhány megtalálható a tervezési osztálydiagramok esetében is finomított formában (metódus,

attribútum), néhány eszköz pedig újonnan jelenik meg (pl. interfészek), mások pedig kiesnek (pl. asszociációs osztályok)

Az elemzési osztálydiagramon az attribútumokat csak nevükkel és esetleg a típusukkal írtuk le. A tervezési osztálydiagramon ez további információkkal bővül:

- láthatóság: korlátozhatjuk az attribútum láthatóságát, elérhetőségét. A láthatóság az objektumorientált paradigma bezárás fogalmát valósítja meg. Azt fejezi ki, hogy az adott attribútum az alkalmazás mely pontjáról érhető el, hivatkozható. A láthatóság különböző értékeinek szintaxisa és jelentése nagyjából megfelel a Java láthatóságainak. A következő láthatósági szintek lehetségesek:
    - (private) privát láthatóság. Ekkor az attribútum csak az adott osztályban használható. Jelölése: -
    - (public) publikus láthatóság. Ekkor az attribútumot látja az összes osztály. Jelölése: +
    - (protected) védett láthatóság. Ekkor az attribútumokhoz csak a leszármazott osztályok férhetnek hozzá. Jelölése: #
    - (package) csomag láthatóság. Ekkor az attribútum az osztályt tartalmazó csomagból hivatkozható. Jelölése: ~
- származtatott attribútumok: Egy törtjel (/) jelöléssel az attribútumot származtatottnak (derived) definiálunk. Az attribútum értéket kaphat valamilyen más attribútumból vagy adatból egy képlet, művelet alapján számítva
- osztályattribútum: azokat az attribútumok, amelyek az osztályhoz és nem példányokhoz tartoznak. Aláhúzással jelöljük őket.
- egyéb tulajdonságok definiálása OCL nyelv segítségével (pl. readOnly)

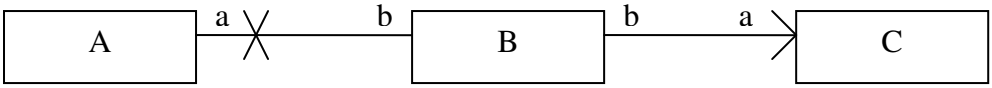


(2.2.1. ábra)

A 2.2.1. ábrán a „Felhasználó” osztálydiagramban megjelentek az új kifejezőeszközök. Láthatjuk, hogy a „jelszó” egy származtatott attribútum, hiszen ennek értékét a „setJelszó” metódus a „felhaszn. név”-ből határozza meg.

Az osztályok közötti kapcsolat fogalma a tervezési osztálydiagramok estében új elemmel bővül. Ez a **navigálhatóság**. Ez nemcsak tartalmi viszonyt fejez ki két osztály között, hanem lehetővé teszi, hogy az asszociáció neve alapján az egyik osztályból az asszociált osztály felé navigálhatunk, vagyis a navigálhatóság annak igényét fejezi ki, hogy az asszociációban résztvevő osztályok példányai a kapcsolaton keresztül navigálva elérhetik a kapcsolódó objektumokat, 2.2.1. ábra:

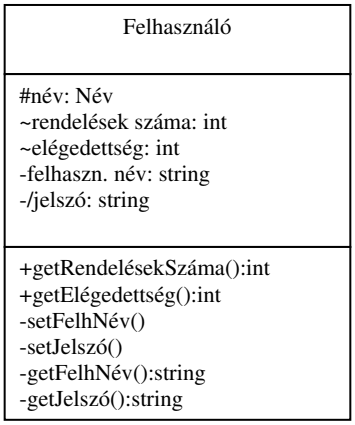
- ha egy asszociáció végén nyílhegy áll, akkor a nyíl irányába navigálhatunk
- ha egy kereszt áll az egyik végén, az azt jelenti, hogy abba az irányba nem lehet navigálni
- ha sem nyíl, sem kereszt nem szerepel, akkor a navigálhatóság nincs specifikálva



(2.2.2. ábra)

A műveletek specifikációja eddig nevekre korlátozódott. Az attribútumokhoz hasonlóan a láthatóság, típus (visszatérési érték) és különböző tulajdonságok is megadhatók:

- A láthatóság az attribútumok láthatóságához hasonlóan azt jelzi, hogy az adott metódus az alkalmazás mely pontjáról hívhatóak, aktiválhatóak. A megadható láthatósági szintek és jelölésük megegyezik az attribútumoknál elmondottakkal
- A paraméterlista segítségével megadhatjuk a művelet mögött álló viselkedés megvalósításához szükséges inputot. Minden egyes paraméterhez megadható a név, az irány, a típus és egy alapértelmezett értek. Az irány a következő értékeket veheti fel:
    - in: a paraméter olvasható
    - out: a paraméter írható
    - inout: a paraméter kiolvasható majd visszaírható
- A visszatérési érték típusa a művelet mögött álló viselkedés outputjának típusát adja meg (return)



(2.2.3. ábra)

A tervezési szakaszban számos újabb eszköz és fogalom jelenik meg. Ezek az eszközök már a probléma megoldásának irányába mutatnak:

- absztrakt osztályok: olyan osztályok, amelyeknek alosztályai lehetnek, de példányai nem
- interfészek: absztrakt osztályhoz hasonlítanak, viszont nem tartalmaznak viselkedést, csak műveleteket, vagyis a metódusoknak csak a specifikációját

2.3 Megvalósítási osztálydiagram

Az előző két fejezetekben (2.1 és 2.2) láthattuk, hogy a modellezést szolgáló kifejezőeszközök egyre inkább a megvalósítás irányába mutatnak. A modellezett probléma egyre pontosabb leírását lehetővé tévő eszközöket használhatunk a szoftverfolyamat egyes fázisaiban. A megvalósítási osztálydiagramban is újabb eszközök jelennek meg, amelyek már inkább a programozáshoz közelítenek, tehát a technológiai megvalósítás áll a középpontban.

Az UML különböző **adattípusoka**t kínál:

- alaptípusok (PrimitiveType): strukturálatlan adattípusok. Az UML-ben a boolean, integer, unlimitednatural és string alaptípusok használhatók
- felsorolásos típusok (Enumeration): egy felhasználó által definiált (egyben strukturált) típus, értékek egy véges, előre megadott halmazával
- olyan összetett adatszerkezet, mint a lista az UML-ben nincsenek, e célokra osztályok szolgálnak

A megvalósítási osztályok közelebb állnak a programozáshoz. Bizonyos megszorítások mellett, illetve átalakítások után az UML megvalósítási osztálydiagramjai Java programoknak tekinthetők (<<Java >> sztereotípiával jelöljük):

- nevek: Java támogatja a Unicode-ot. Kevés korlátozás van az azonosítók nevére vonatkozóan
- adattípusok: adattípusként használhatók a Java alaptípusai
- osztálynak a class, absztrakt osztályt abstract class, interfészt interface, a csomagot pedig package kulcsszóval vezetünk be
- stb.

Így a „Felhasználó” osztály java kódja:

```
public class Felhasznolo{
        protected Nev nev;
        int rendelesekSzama;
        int elegedettseg;
        private string felhasznNev;
        private string jelszo;
        public int getRendelésekSzáma(){}
        public int getElégedettség(){}
        private void setFelhNév(){}
        private void setJelszó(){}
        private string getFelhNév(){}
        private string getJelszó(){}
        }
```

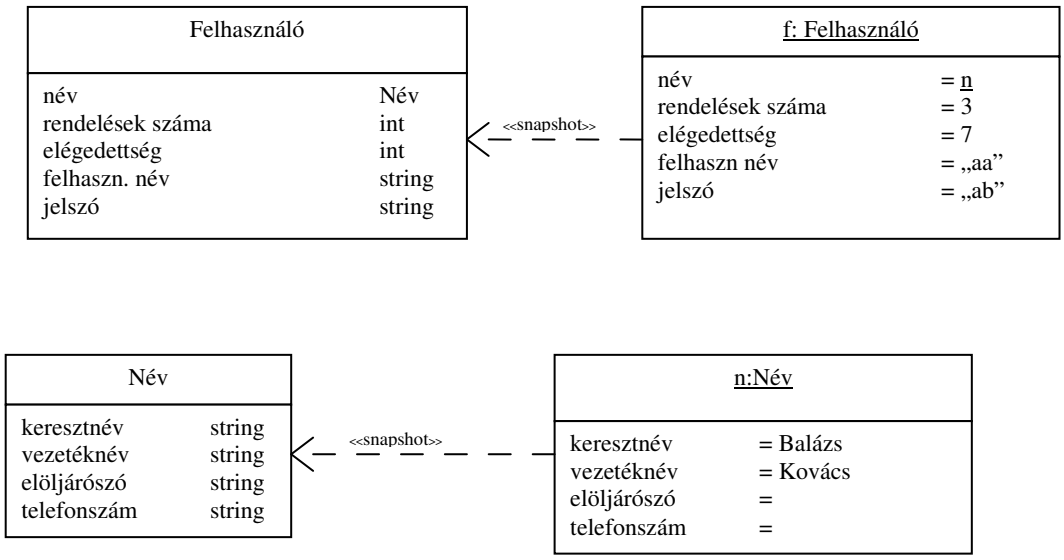
2.4 Objektumdiagram

Az objektumdiagramok segítségével a rendszert alkotó objektumok egy adott időpillanatban felvett állapotát modellezhetjük. A diagram segítségével konkrét helyzeteket, vagy ennél absztraktabban jellemző helyzeteket is felvázolhatunk. Az objektumdiagram kiválóan alkalmas az osztálydiagramok tesztelésére, annak ellenőrzésére, hogy konkrét adatok, kapcsolatok esetén helytálló a modellünk. Az objektumdiagram szintén alkalmas egyes rendszerfolyamatok nyomkövetésére, illetve nem kívánt állapotok feltárására.

Egy objektum jelölése -az osztálydiagramokhoz hasonlóan- téglalappal történik. A név rekeszébe az objektum neve kerül aláhúzva. Az objektum neve két részből áll, melyeket kettősponttal választunk el egymástól:

- objektumot azonosító név
- objektum osztályának neve

Szemléltethetünk anonim objektumokat is. Ekkor az objektum nevét elhagyjuk, és csak a kettőspontot, valamint az osztály nevét tüntetjük fel. Az anonim objektummal azt fejezhetjük ki, hogy az osztály minden példányára érvényes az adott körülmény.
Az osztályok attribútumokkal rendelkeznek, és az osztály példányai ezen attribútumokra értékeket vesznek fel. Az attribútumértékek határozzák meg az adott objektum állapotát. Az objektum által felvett értékeket az objektumdiagram névrésze alatt, egy külön rekeszben tüntetjük fel: az attribútum neve után megadjuk a felvett értéket (nem kötelező minden attribútumértéket megadni).
Mivel az osztályok között kapcsolat áll fenn, ezért az osztályok példányai között is kapcsolat van. Az objektumok között fennálló kapcsolatokat hasonlóan az osztályok közötti asszociációkhoz az objektumok között húzott vonallal jelölhetjük.



(2.4.1. ábra)

3 ARCHITEKTÚRA ÉS KOMPONENSEK

A szoftver architektúra:

- a rendszert felépítő alrendszerek (komponensek) kerete
- strukturálisan fontos elemek, és azok kapcsolata határozzák meg az alkalmazás felépítését
- a rendszer alapvető komponenseinek szerveződése, melyek egymással meghatározott felületeken keresztül kommunikálnak

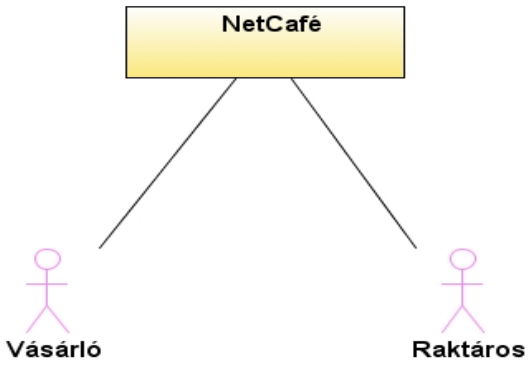
Az alkalmazás fejlesztése során lényeges, hogy megfelelő struktúrát alakítsunk ki, amely jelentősen befolyásolja a szoftver átláthatóságát, használhatóságát. A modularitásnak köszönhetően az egyes komponensek újrafelhasználása válik lehetővé. Az architektárális tervezésnek köszönhetően:

- átláthatóvá válik a fejlesztés
- könnyebben felismerhetők az újrafelhasználható elemek
- átlátható a projektmenedzsment
- a kockázatok csökkennek
- lehetővé válik a párhuzamos fejlesztés

3.1 Kontextusdiagram

A kontextusdiagramok egy rendszer, illetve annak környezetének kijelölésére és az azzal kapcsolatban lévő résztvevőkkel történő kommunikáció szemléltetésére szolgál. A kontextusdiagramok már a 80-as években jelen voltak a strukturális módszertanok jegyében. A diagramon az egyes rendszerelemeket téglalapban tüntetjük fel. A rendszer egyes elemeihez más rendszerelemek kapcsolódhatnak: ezt az asszociációhoz hasonlóan vonallal jelöljük. Az alkalmazás használata során a felhasználó kapcsolatba lép a szoftver egyes

elemeivel, illetve magával a rendszerrel. Az UML terminológiájában a pálcafigurákat **aktornak** (actor) hívják. Egy aktor mindig egy szerepkört képvisel, nem pedig egy konkrét személyt. Az osztályokkal ellentétben az aktorok nem részletezhetők, és a viselkedésük sincs modellezve a diagramon. Egy vonallal jelezzük az aktor és a rendszer közötti interakciót.



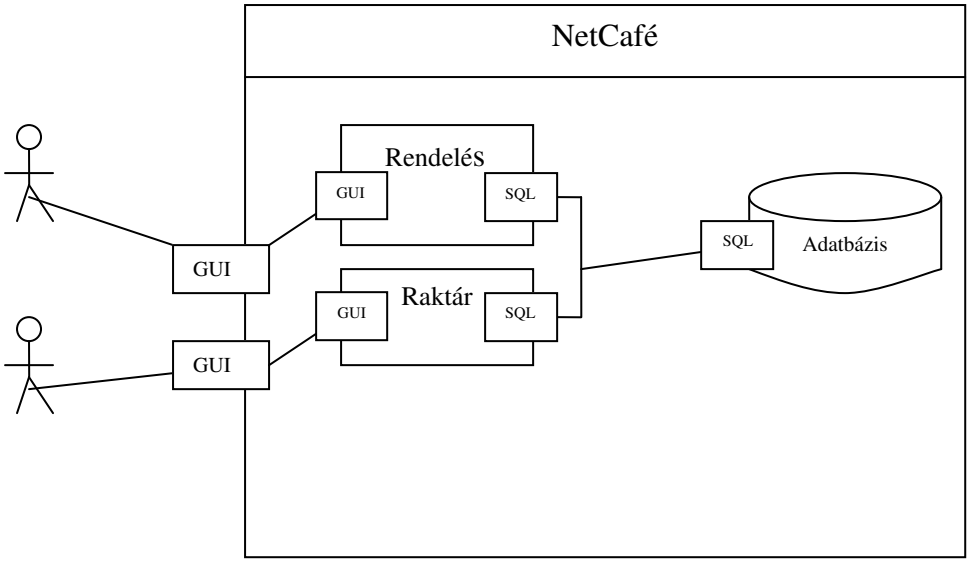
(3.1.1. ábra)

A 3.1.1. ábrán láthatjuk, hogy a „NetCafé” rendszerrel 2 aktor áll interakcióban:

- a vásárló: az a személy, aki a kávéház által kívánt termékek vásárlása céljából használja a rendszert
- a raktáros: az a személy, aki a raktáron lévő termékek romlandóságát ellenőrzi. Termékeket rendel, vagy selejtez

3.2 Montázsgiagramok

Ebben a fejezetben a struktúra fogalom alatt az összetett struktúrát fogom érteni: egymáshoz kapcsolódó futási idejű elemek, amelyek egy cél elérése érdekében működnek együtt. A montázsdiagramok egy rendszer, illetve összetettebb osztályok felépítését írják le, így architektúradiagramnak is nevezik őket.



(3.2.1. ábra)

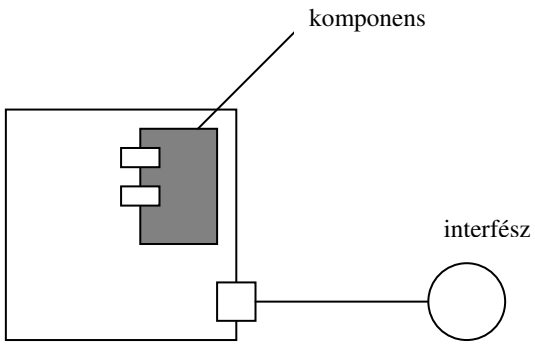
A kontextusdiagramból kiindulva megkonstruálhatjuk a **rendszer-montázsdiagramját.** Itt a „NetCafé” rendszer struktúrája részekkel, csatlakozókkal és összekötőkkel van finomítva:

- a kontextusdiagram finomításaként a „NetCafé” rendszert további részekre bontjuk, amelyek mindegyike egy funkcionalitást hordoz önmagában. A részeket egy önálló rendszerként realizáljuk („Rendelés”, ”Raktár”). Emellett a rendszerben megjelenik egy adatbázis, amelyet a két alrendszer közösen használ. Nyilvánvalóan ezek a részek tovább finomíthatóak
- a csatlakozók (port) kis fehér, feliratozott négyzetekként vannak ábrázolva. A csatlakozó egy osztály interakciós pontja, amely arra szolgál, hogy az osztállyal kapcsolatos összes előforduló interakciót megvalósítsa. A csatlakozók az osztály bezárását valósítják meg. Mivel a portokon keresztül valósul meg a kommunikáció, így az osztály belseje rejtve marad. Azt, hogy melyik osztályhoz tartoznak, az osztály keretén elhelyezett kis négyzet jelöli. Három csatlakozótípust különböztetünk meg:
    - belső csatlakozó: a rendszer részeinek kapcsolatát megvalósító csatlakozók
    - relécsatlakozó: a belső csatlakozók elrejtését valósítja meg
    - transzpondercsatlakozó: funkciót lát el (pufferelés, jelek átkódolása)

- az egységbezárás biztosítása végett a részek kapcsolódása csak csatlakozókon keresztül valósul meg, amelyek egymáshoz összekötőn (connector) keresztül kapcsolódnak. Az összekötők a montázsdiagramon felirat nélküli folytonos vonalként jelennek meg

3.3 Komponensdiagram

Az UML-komponens egy egységbezárt, önálló, teljes és ezáltal cserélhető egység, amely függetlenül működtethető, telepíthető és összekapcsolható más komponensekkel. A komponens az UML-metamodellben osztályok (<<Component>> sztereotípiával adható meg) egy alosztálya, tehát rendelkezik az összes jellemzővel: a komponenseknek is vannak attribútumai, metódusai, csatlakozói és így tovább. A komponenseknek lehet csatlakozói, amelyek interfészekkel írhatók le:

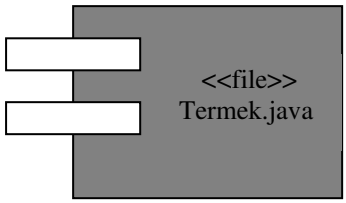


(3.3.1. ábra)

A komponenseket a diagramon egy téglalappal jelölhetjük, amelynek bal oldalán két téglalap alakú címkét veszünk fel. A komponenseket névvel láthatjuk el, amelyet a téglalapba írunk. A komponens alkalmazás architektúrájában betöltött szerepét szintén a téglalapba írva

sztereotípiával fejezhetjük ki. Az UML által előredefiniált, komponensekhez rendelhető sztereotípiák:

- <<executable>>: futtatható programot tartalmazó állományt jelöl
- <<library>>: programkönyvtár, melyre egy program hivatkozik
- <<table>>: adatbázistábla
- <<file>>: forráskód, vagy adatot tartalmazó állomány
- <<document>>: dokumentumot tartalmazó állományt jelöli



(3.3.2. ábra)

A komponenseknek különböző fajtái vannak:

- futási idejű komponensek: funkcionálisan kicsi, futási időben létező egység
- tervezési komponensek: a szoftverfejlesztés fontos építőelemei, a komponensek különböző szemléletmódjait kapcsolják össze
- megvalósítási komponensek
- alrendszerek: nagyméretű szakmai egységet zárnak be

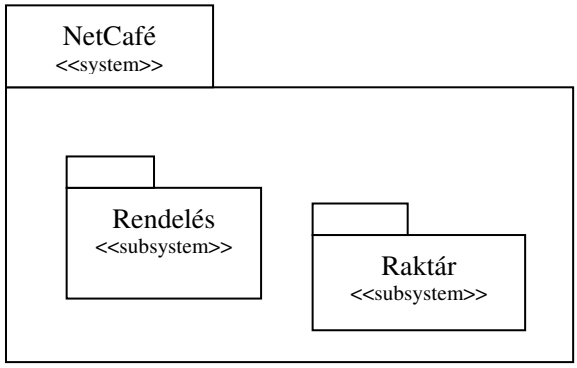
A komponensek alkalmasak az alkalmazás fizikai szerkezetének szemléltetésére. Így a rendszert alkotó fizikai szoftvermodulok és azok kapcsolódásának modellezésére használhatóak. A komponensdiagram az osztálydiagram osztályainak és egyéb elemeinek fizikai csoportosítását ábrázoló diagram: szoftver-komponenseket, interfészeket és ezek közötti kapcsolatokat tartalmaz. A szoftver-komponensek programállományok (forráskódok), bináris állományok és végrehajtható programok lehetnek.

3.4 Csomagdiagram

A csomag elemek csoportosítására, közös névtérben való elhelyezésére, egyesítésére szolgál. Így csomagok segítségével a modellünk szorosabban összetartozó elemeit csoportosíthatjuk, magasabb szintű egységekbe szervezhetjük. A csomag különböző típusú elemek csoportosítását valósíthatja meg:

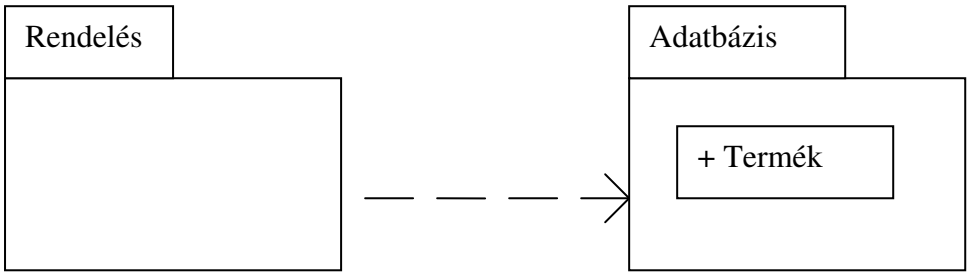
- osztályok
- interfészek
- komponensek
- diagramok
- egyéb elemek

A csomag névterületet is jelent, amelyen belül az egyes elemek elnevezésének egyedinek kell lenni. A modell minden eleme pontosan egy csomaghoz tartozik, így a tartalmazási hierarchia egy fát alkot. A csomagot grafikusan egy téglalappal jelöljük, amelynek bal felső sarkára elhelyezett kisebb téglalap kerül (függő mappa). A csomag nevét magára a mappára írjuk. A csomagdiagram egy gráf, amelyben a gráf csomópontjai maguk a csomagok, élei pedig a csomagok közötti kapcsolatot, függőségi viszonyt reprezentálják. A csomagokon belül további csomagok lehetnek, tehát a csomagok egymásba ágyazhatók.



(3.4.1. ábra)

A csomagbeli elemekhez is rendelhetünk láthatóságot. Az elem neve előtt helyezhető el (osztályoknál leírtak), viszont csak public és private értékek adhatók meg. Ha nincs megadva láthatóság, akkor az adott elem nyilvánosnak számít. A csomagon belül levő elemek egymásra közvetlenül hivatkozhatnak, a csomagon kívül eső elemeket pedig minősítéssel érhetik el. A csomag összes nyilvános eleme minden más névtérből elérhető teljes minősített név segítségével. Ez nehéz kezelhetőséget teremt. A teljes minősített névvel való hivatkozás nehézségét megoldhatjuk, ha a hivatkozó csomagban **importkapcsolatot** alakítunk ki a hivatkozott csomaggal. Az importkapcsolatot szaggatott, nyitott hegyű nyíllal jelöljük.



(3.4.2. ábra)

Két különböző függőséget különböztetünk meg:

- egyedi import: csomag elemeit külön-külön importáljuk
- csomagimport: a csomag összes elemét importáljuk

Sztereotípiák használatával a csomagok típusát és sajátságait is meg tudjuk adni:

- <<system>> a hierarchia tetején álló legfelsőbb csomag (teljes alkalmazás)
- <<subsystem>> alkalmazáson belüli csomag
- <<facade>> már modellezett csomag eltérő nézetben való megjelenítése
- <<import>> importkapcsolatot jelöl ki
- egyéb

4 VISELKEDÉSI DIAGRAMOK

4.1 Használati esetek

4.1.1 Funkcionalitások, követelmények

Az informatikai rendszerek mindig valamilyen célt szolgálnak, valamilyen probléma megoldására születnek meg. Ahhoz, hogy az adott alkalmazás betöltse és kielégítse a kívánt funkciókat, a követelmények meghatározása elengedhetetlen. A követelményekben fogalmazódnak meg az alkalmazással szembeni elvárások, illetve azok az egyedi igények, amelyek az adott szakterülettel kapcsolatban merülnek fel. A használati eset diagram a rendszer felhasználóinak a rendszerrel szemben támasztott elvárásait, követelményeit modellezi. A követelmények lehetnek:

- nemfunkcionális követelmények
- funkcionális: a rendszer vonatkozásában beszélhetünk folyamatokról. A folyamatok egyes lépései használati esetek segítségével írhatók le

Az alkalmazással szemben támasztott követelmények összegyűjtésének, és a használati eset diagram kialakításának legfontosabb lépései:

- a rendszerhez kapcsolódó aktorok (szereplők) összegyűjtése
- a rendszer működését, folyamatait kifejező használati esetek összegyűjtése
- az aktorok és használati esetek kapcsolatainak meghatározása

A használati eset diagram az összegyűjtött követelmények megvalósításáról, implementációjáról semmiféle információt nem tartalmaz. A használati eset diagram modellezi a rendszer vagy annak egy részének viselkedését. Inkább egy külső nézetet ad, amiben azonosíthatók a követelményekhez kapcsolódó folyamatok, valamint a rendszerhez kapcsolódó külső elemek is.

4.1.2 Folyamatok, aktorok

Az alkalmazás célja, hogy az adott problémára megoldást nyújtson, és az ehhez szükséges folyamatokat realizálja. A kontextusdiagramból kiindulva a rendszer működésének határai meghatározhatók, azonosítanunk kell a rendszer vonatkozásában azokat a folyamatokat, melyeket használati esetként használunk fel. Az **aktor** egy szerepkört fejez ki, azaz a rendszerhez kapcsolódó elemek egy típusát azonosítja. Aktor lehet:

- felhasználó
- hardvereszköz
- másik alkalmazás

Az aktorokat általában egy pálcikaemberrel jelöljük, és alá írjuk az adott aktor elnevezését.

(4.1.2.1. ábra)

Egyfajta szerepkör-hozzárendelést kell elvégeznünk, vagyis meghatározzuk, hogy a rendszeren kívül eső elemek, vagyis az aktorok mely folyamatokhoz kapcsolódnak: az

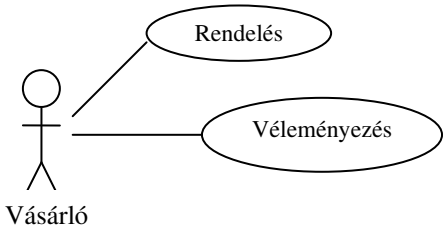
aktorok általában különböző céllal kapcsolódnak a rendszerhez, annak valamilyen funkcióját szeretné elérni, használni. Ezeket a funkciókat **használati esetekkel** írjuk le. A használati esetek a rendszer azon fontos viselkedésmódját hordozza magában, amelyek:

- hiánya a rendszerrel szemben támasztott követelmények kielégítetlenségét okozza
- az aktor számára valamilyen eredményt szolgál

A használati eseteket ellipszissel jelöljük, amelybe beleírjuk a használati eset nevét. Fontos, hogy a használati eset neve ne legyen semmit mondó, mivel a használati esettel reprezentálunk egy folyamatot, vagy egy folyamat egy részét.

(4.1.2.2. ábra)

Az aktorok és a használati esetek kapcsolatát asszociációval fejezzük ki: az aktor és a használati eset kommunikációját fejezi ki.

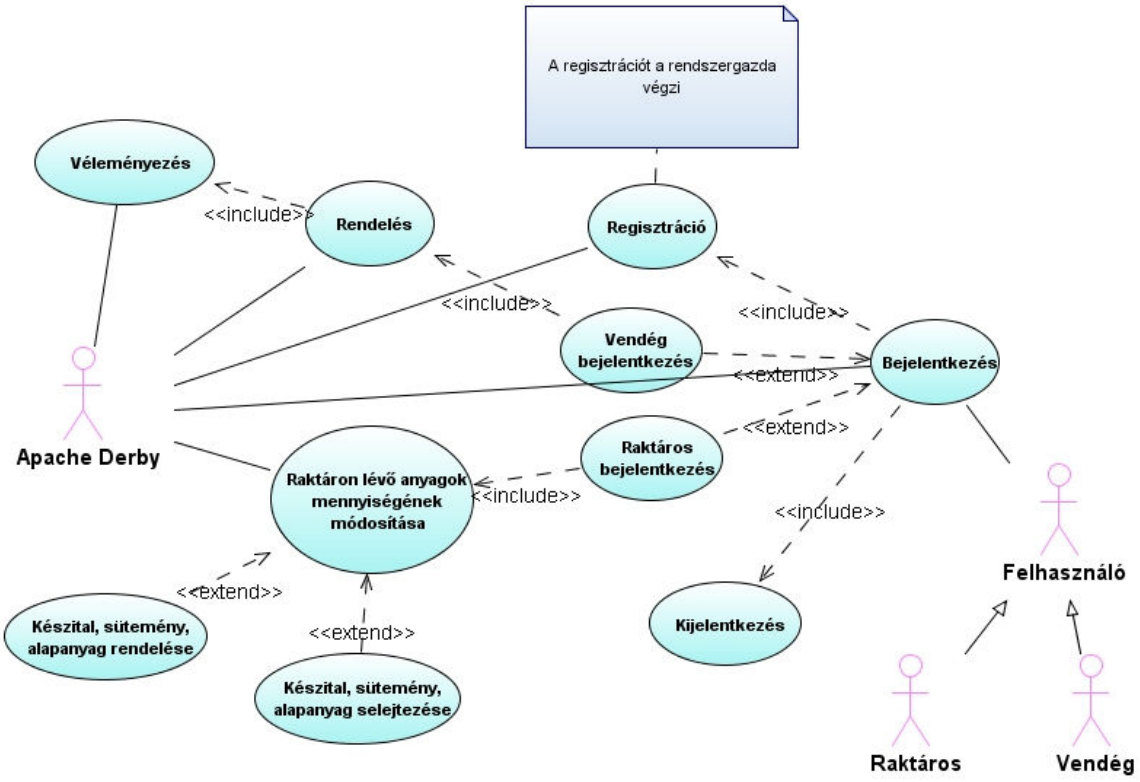


(4.1.2.3. ábra)

4.1.3 Függőségek, kapcsolatok típusa

Informatikai rendszerek esetén általában nagy számban találhatunk olyan folyamatokat, használati eseteket, amelyek összefüggésben állnak egymással. Különböző függőségi, kapcsolati viszony lehet ezek között:

- magábafoglalás: <<include>> tipusú kapcsolattal egy használati eset vonatkozásában egy részfunkció kiemelését reprezentálhatom, valamint a folyamatok (használati esetek) időbeli sorrendiségét is kijelölhetem
- kiterjesztés: <<extend>> tipusú kapcsolat egy alap használati eset viselkedésének kiterjesztésére, kibővítésére szolgál
- általánosítás/pontosítás (generalization): aktorok és használati esetek kapcsán is értelmezhető, jelentése megegyezik az objektumorientált paradigma fogalmával.
    - aktoroknak lehetnek alváltozatai. A közös tulajdonságot kiemelhetjük egy közös ősbe, a leszármazott aktorok ugyanazokat a használati eseteket kezdeményezhetik, mint az ősük
    - használati esetek között is lehetnek hasonlóságok (szerkezet, viselkedés). A közös részeket kiemelhetjük egy általános használati esetbe. A leszármazottak öröklik az ősük struktúráját, viselkedését és kapcsolatait, ezeket felüldefiniálhatják, és további viselkedéseket is deklarálhatunk



(4.1.3.1. ábra)

4.2 Tevékenységek

A **tevékenységdiagramo**k felhasználási köre nagyon széles, a legkülönbözőbb folyamatok szemléltetésére használhatjuk. A tevékenységek magukban hordozzák a rendszer lényeges jellemzőit, funkcióit, így az alkalmazásban lezajlódó folyamatokat felépítő lépések meghatározása elengedhetetlen. A tevékenységdiagramok hosszú múltra tekintenek vissza:

- legelső előfutára valószínűleg a struktogram és a Nassi-Sneiderman diagram. Mind a kettőt főleg programok lefutásának leírására használták
- A 70-es évek közepén jelent meg az adatfolyam-diagram (data flow diagram), ami a rendszer adatáramlási folyamatainak szemléletes ábrázolását szolgálta, valamint az IR

funkcionális jellegét hangsúlyozta, vagyis hogy a fejlesztő ne rögtön a programozással foglalkozzon. Az adatfolyam-diagram már magasabb absztrakciós szintet valósít meg

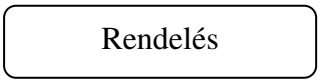
- IDEF-3, SAP/R3

A diagram az alkalmazás dinamikus viselkedését, a rendszerben lefutó folyamatok, tevékenységek vezérlését, sorrendiségét modellezi. Mivel a diagram absztrakciós szintje elég magas, ezért a konkrét megvalósításról nem nagyon mond semmit, bár vannak a programozás irányába mutató eszközök. Tehát, ha valamilyen folyamat végrehajtást, lefutást szeretnénk szemléltetni, akkor a tevékenységdiagramot használhatjuk.
A tevékenységdiagram a használati eset diagramhoz hasonlóan különböző elemekből épül föl:

- a folyamat kiindulópontját is külön jellel jelöljük, valamint a végállapotát (termination node), amely a folyamat végét jelzi

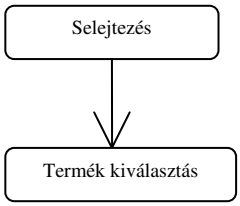
(4.2.1. ábra)

- a diagram alap építőeleme egy tevékenységet szimbolizáló ívelt oldalú téglalap. A téglalapba írjuk az adott aktivitás nevét, amely önmagában is információt hordoz arról, hogy milyen cselekvést, feladatot kell a folyamat adott pontján végrehajtani. Egy tevékenység tetszőleges szinten jelenhet meg, azaz rendszerint nem atomi lépést szemléltet, így általában a tevékenységek is részletezhetők



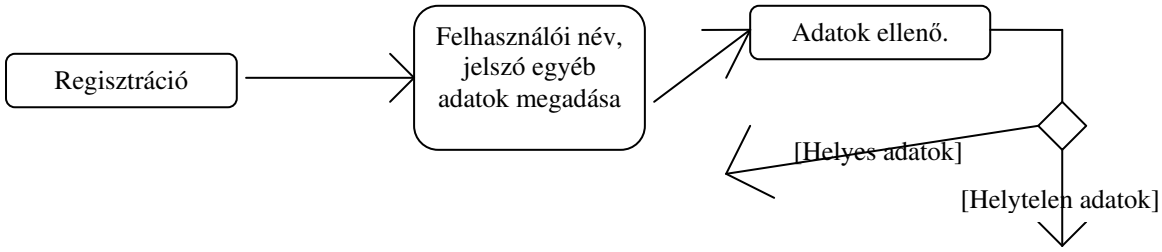
(4.2.2. ábra)

- a tevékenységdiagram középpontjában az átmentek állnak. Egyik tevékenységből a másik tevékenységbe való jutását a diagramon egy irányított nyíllal jelöljük, amely a megelőző aktivitásból mutat a rákövetkező aktivitásra. Tehát az átmenetekkel sorrendiség is definiálható a tevékenységek között



(4.2.3. ábra)

- a tevékenységek végrehajtása nem miden esetben egymás utáni sorrendiséget tükröz. Előfordulhat, hogy egy tevékenység végrehajtását valamilyen feltételhez kötjük, a feltétel nem teljesülése esetén esetleg más tevékenység végrehajtása kerül sorra. A tevékenységdiagramon ezt úgynevezett őrszemmel (megszorítással- permission) tehetjük meg. Így ha több feltételt is megadunk, akkor az esetválasztó (choice pseudosate) csúcs segítségével, amit egy rombusszal jelölünk, az adott feltételnek eleget tevő átmenetbe juthatunk. Azokon az átmeneteken, melyek az esetválasztó csúcsból indulnak ki, szögletes zárójelben feltüntetett feltételek szerepelnek

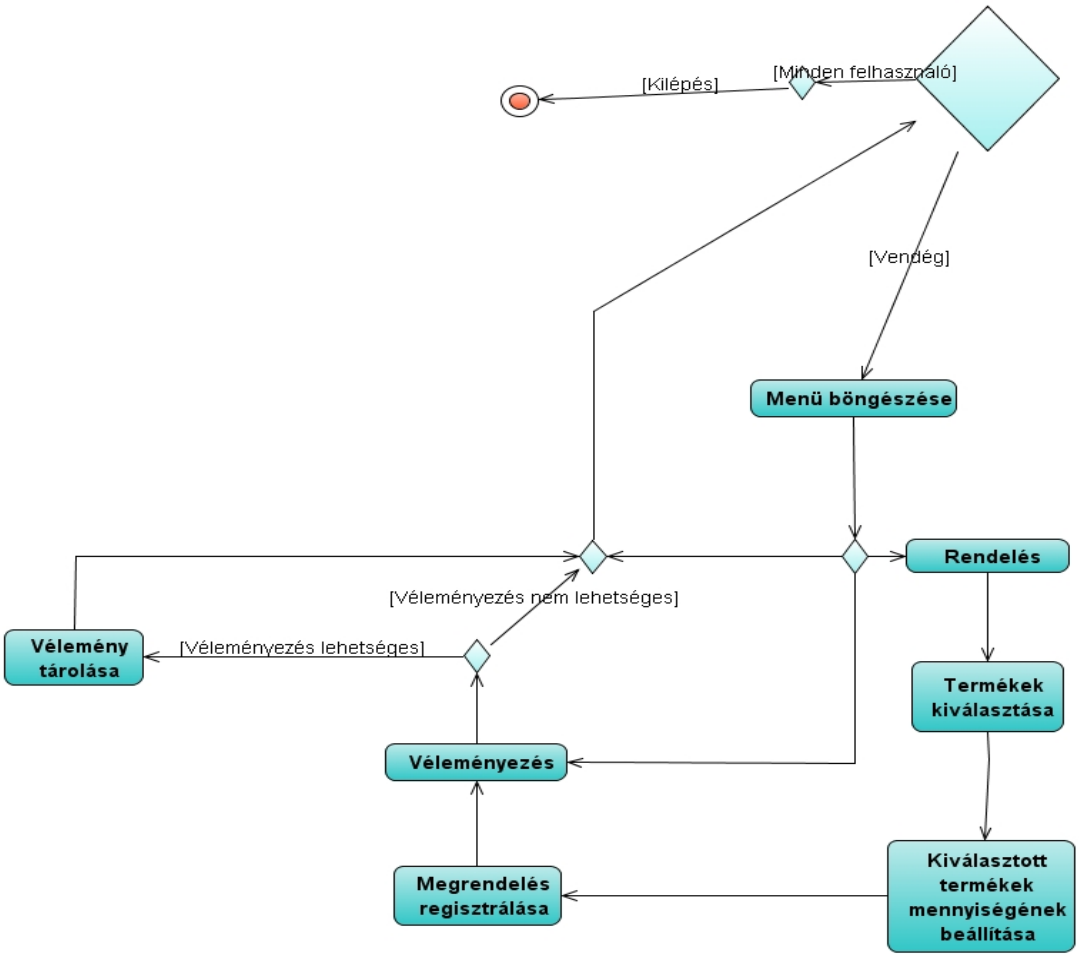


(4.2.4. ábra)

- az UML tevékenységdiagramján lehetőségünk van párhuzamos végrehajtás leírására is. Ezt vezérlés elválasztó csomópont (Forknode) segítségével tehetjük meg, amellyel szemléltetjük, hogy a folyamat végrehajtása az adott ponttól két (vagy több) független, párhuzamos szálra bomlik. A vezérlési szálak egybeszövődése egy összeolvasztó csomópont (Joinnode) használatával történik meg

(4.2.5. ábra)

- előfordulhat, hogy bizonyos tevékenységek többszöri, egymás utáni végrehajtását szeretnénk reprezentálni. Erre az UML a ciklusszervező csomópont (Loopnode) eszközt nyújtja. A ciklusszervező csomópont három részből áll: inicializáló, törzs, ellenőrzés



(4.2.6. ábra)

A 4.2.6. ábra a „NetCafé” rendszer a vásárlás folyamatát modellezi.

A tevékenységdiagramok felhasználási módja nagyon széles, és különböző nézetekből írhatjuk le a rendszerben zajló folyamatokat. Az UML 2.0 verziója új jelöléseket vezet be:

- kivételeket szemléltethetünk. Kivétel váltódhat ki, ha például egy feldolgozási hiba lép fel. Ekkor az egész tevékenységet meg kell szakítani, hogy a hiba kezelése a tevékenységen kívül végbemehessen. Ha egy csomóponton felkészülünk a kivételre, akkor védett csomópontról beszélhetünk. A kivételt ki kell váltani, és el kell kapni,

valamint kezelni kell. A kivétel egy külön vezérlési ágat indít el, a keletkezett kivétel objektum egy kivételkezelő csomóponthoz kerül, ami a kivételt lekezeli

- megszakítási területek. Alaphelyzetben a kivétel hatásköre a teljes tevékenység, ami azt jelenti, hogy az egész tevékenység befejeződik. Ez hatáskör megszakítási terület alkalmazásával korlátozható. A diagramon ezt a tevékenység területét körülhatároló szaggatott vonallal jelezzük. Ha egy megszakítási területen kivétel váltódik ki, akkor nem az egész tevékenység fog befejeződni, hanem csak az adott terület vezérlési folyamata

4.3 Interakciók

Az objektumorientált világban a rendszert alkotó objektumok rendszerint nem önmagukban léteznek, hanem kölcsönhatásban vannak más objektumokkal. Egy OO-t tükröző alkalmazásban az objektumok a kapcsolatokon keresztül kommunikálnak: valamilyen szolgáltatást kérnek, vagy esetleg szolgáltatást nyújtanak. A bonyolultabb viselkedés leírására született meg ez a diagramtípus.

Az interakciós diagram alapvetően a rendszerben végbemenő üzenetváltások leírására, egységek közti információcsere jelölésére szolgál. Az üzenetváltás több kölcsönhatásban lévő résztvevő között zajlik le. Résztvevő lehet például egy osztály, objektum, aktor, komponens, csatlakozó.

Az interakciós diagramok továbbá használhatóak követelmények, illetve tesztesetek leírására, de általában az alkalmazás viselkedését specifikáljuk segítségükkel.

Az UML jelenleg az interakciós diagramok három komplementer változatát támogatja, ezek csak jelölésbeli eltérésekben különböznek: szekvenciadiagram, idődiagram és kommunikációs diagram.

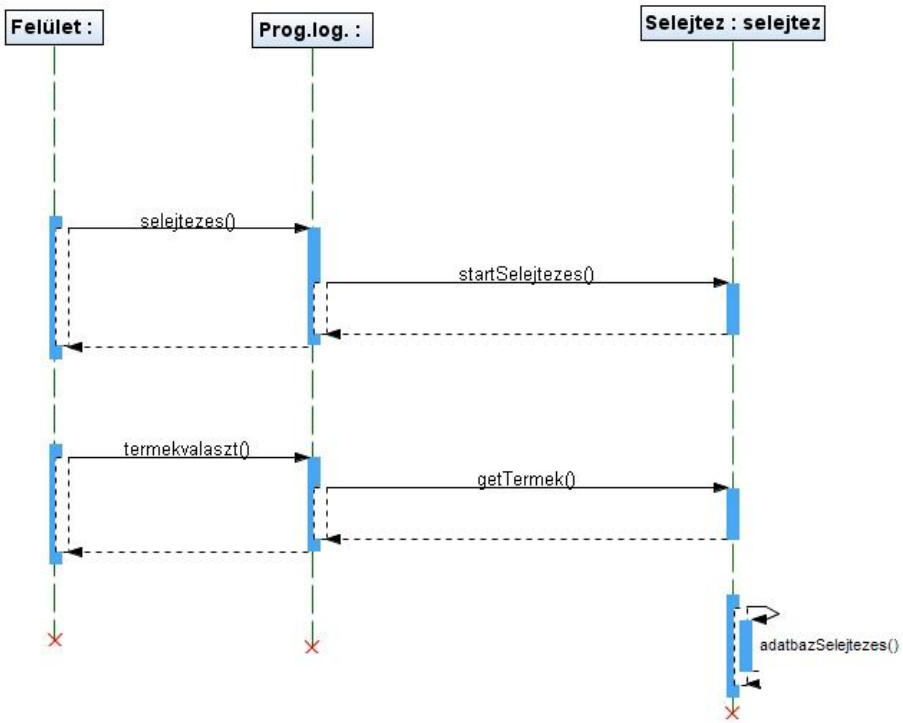
Szekvenciadiagram

A kölcsönhatásban lévő résztvevőket az objektumdiagramban leírt jelöléssel, és abból kiinduló szaggatott vonallal (életvonal) jelöljük, ami időrendiséget követ felülről lefelé nézve. Az életvonalak mentén esemény előfordulásokat találunk, például üzenetküldést vagy metódushívást. Az esemény előfordulást az interakciós résztvevők közötti üzenetváltási nyíl

jelzi, amely a küldő életvonalából mutat a fogadó életvonalába, és amely az üzenet nevét is hordozza. Különböző típusú üzenetküldéseket különböztetünk meg:

- szinkron üzenet esetén a küldő mindaddig nem kezdhet bele új tevékenység végrehajtásába, míg a fogadó partner fel nem dolgozza a kérést, és vissza nem küldi a választ, amellyel együtt a vezérlés is visszakerül a küldőhöz. Szinkron üzenetváltást telt fejű nyíllal jelöljük. Mivel szinkron üzenetről van szó, ezért a küldő vár a fogadó válaszára, amelyet a fogadó életvonalából a küldő életvonalába húzott szaggatott vonalú nyíllal jelzünk
- aszinkron üzenet esetén a küldőnek nem kell válaszra várnia, csak annyi a feladata, hogy üzeneteket küldjön. Az aszinkron üzenetváltás nyitott végű nyíllal jelöljük
- feltételes üzenetküldés esetén az üzenetküldést szimbolizáló nyílon szereplő név előtt szögletes zárójelben elhelyezett logikai kifejezés áll
- amikor egy fogadónak többszöri üzenetküldést kell megvalósítania, akkor ezt iterációnak nevezzük. Ezt az üzenet neve előtt elhelyezett csillaggal, illetve a végrehajtás számát kifejező értékkel jelöljük

Előfordulhat, hogy egy interakciós résztvevőnek saját műveletét kell meghívnia. Ezt öndelegációnak nevezzük, és ekkor az interakciót jelző nyíl az adott életvonalból indul, és oda is tér vissza. Az esemény előfordulások elrendezése az életvonal mentén a sorrendjüket reprezentálja, viszont a köztük lévő távolság semmilyen időmennyiséget nem fejez ki. Az életvonalakat aktív és passzív szakaszra oszthatjuk. Addig, míg egy interakciós partner aktív, az életvonalát egy aktivitási sáv (activation bar) foglalja el. Az aktivitási szál addig a pontig tart, ameddig az aktivált állapot véget ér. Ha egy résztvevő aktiválása felfüggesztődik, de nem szakad meg, azt szaggatott vagy szűkebb aktiválási sávval jelöljük. Ez például akkor fordul elő, ha egy küldő partner vár a fogadó válaszára. Az üzenetek futási ideje függ a szinkronizációtól. Az elhanyagolható futási idejű üzenetek ábrázolása vízszintes nyíllal jelöljük. A számottevő futási idejű üzenetek ferdén lefelé mutató nyíllal kerülnek ábrázolásra. A diagramon azt, hogy egy interakciós partner tovább nem vesz részt a kölcsönhatásban, azt az életvonalon elhelyezett X-szel jelöljük. A szekvenciadiagramok akkor használhatóak jól, ha az interakció kevés partner között zajlik, és ezek sok üzenetet cserélnek.



(4.3.1. ábra)

A **kommunikációs diagramon** az üzenetek sorrendjét csak számozásukkal tudjuk meghatározni. Így a kommunikációs diagramok akkor használhatóak jól, ha az üzenetváltások száma nem túl nagy, mivel az üzenetek számának növekedésével a diagram áttekinthetetlenné válik.

Az **idődiagram** függőleges tengelyén az interakciós résztvevők vannak, esetleg néhány állapotukkal. A vízszintes tengely a szekvenciadiagram függőleges tengelyével ellentétben valódi időt reprezentáló tengely, vagyis a vízszintesen mért távolságok egyenesen arányosak az időbeli távolságokkal.

5 ÖSSZEFOGLALÁS

A szakdolgozatom célja az volt, hogy bemutassam az UML gyakorlati alkalmazásának lehetséges módjait. Az UML folyamatosan fejlődik, és egyre szélesebb körű eszközöket nyújt. Olyan általános segítséget nyújt a fejlesztőnek, amely megkönnyíti a szoftverfejlesztés folyamatát. Az UML integrálja azokat a több évtizede létező eszközöket, amelyek a rendszer különböző szemléletű vizsgálatát, szemléltetését teszik lehetővé, ezáltal a szoftveréletciklus bármely szakaszában hasznos segítőtársunk lehet.

A dolgozatomban nem mutattam be az összes eszköz alkalmazását, és azokat sem a teljesség igényével. Sajnos idő hiányában csak a számomra fontos eszközöket mutattam be. Remélem, hogy az általam választott példa alapján sikerült bemutatnom az egyes diagramtípusok alkalmazásának módját és lényegét.

Szeretnék köszönetet mondani témavezetőmnek Pánovics Jánosnak, az egyetemi éveim alatt és a diplomamunkám elkészítése során nyújtott türelméért és segítségéért.

6 IRODALOMJEGYZÉK

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

Vég Csaba-Dr. Juhász István: Objektumorientált világ, IQSOFT, 1998

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

http://www.rational.hu/uml

http://www.uml.com