Debreceni Egyetem
Informatikai Kar

Az UML eszközeinek bemutatása egy komplex rendszer tervezésén keresztül

Témavezető:
Pánovics János
számítástechnikai munkatárs

Készítette:
Grépály Csaba János
V. programtervező matematikus

Debrecen, 2007

Tartalomjegyzék

1 Bevezetés

1.1 Mi az UML ?

A Unified Modeling Language(UML) egy objektum-orientált szemléletre épülő elemző- és tervezőeszköz. Kiterjedt eszközrendszere lehetővé teszi a különböző objektum-orientált koncepciók, fogalmak jelölését, de ugyanakkor használható struktúrált módszerrel elkészítetett modellek ábrázolására is. Közvetlen örököse a korábbi három legnépszerűbb objektum-orientált módszertan, a Booch-módszer, a Rumbaugh és társai által összeállított OMT, valamint a Jacobson-féle OOSE jelöléseinek, közvetlenül ennek a három módszertannak és a hozzájuk tartozó jelölésrendszernek az összevont és letisztított változata. Az UML egy grafikus eszköz, azaz a modellt különböző diagramok segítségével ábrázolja. Az UML ugyanakkor egy nyelv, azaz szintatktikai és szemantikai szabályok összessége. A szintatktikai szabályok a szimbólumrendszert, azok formáját, és kapcsolódási módjaikat definiálják, a szemantikai szabályok pedig az egyes szimbólumok, illetve a szimbólumok kapcsolatainak értelmezését definiálják. Az UML egy modellező nyelv, ahol a modellező jelző arra utal, hogy segítségével csupán az alkalmazás vázlatát készíthetjük el, a teljes implementációt nem.

Az UML egy olyan eszköz, amelynek segítségével a szoftverrendszerek fejlesztői, tervezői specifikálhatják, vizuálizálhatják, és dokumentálhatják a fejlesztett szoftverrendszerek modelljét, egy kész, rendelkezésre álló szabványt nyújt a szoftveripar szereplőinek. Ezen túlmenően az UML az egységesítésnek köszönhetően segíti az alkalmazás tervezői, fejlesztői közötti kommunikációt, kiválóan megfelelő az információcsere eszközének.

1.2 Az UML története

Az UML vázlatos története:

- 1970-es évek közepe: az első objektum-orientált modellezőnyelvek megjelenése
- 1989-1994: ebben az időszakban egyre nagyobb igény jelentkezik az objektum-orientált modellezőnyelvek iránt, a modellezőnyelvek száma megsokszorozódik
- 1994: Grady Booch és James Rumbaugh a nyelvek egyesítésének céljából megkezdik az egységes modellezőnyelv kidolgozását, melyet Unified Method-nak neveznek el
- 1995: csatlakozik a társasághoz Ivar Jacobson, elkészül az egységesített modellezőnyelv első, 0.8-as verziója, immár Unified Modeling Language néven
- 1996: elkészül az UML 0.9-es verziójának specifikációja, 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

1.3 A tervezett rendszer

A tervezett rendszer célja, hogy webes megjelenítési felületet biztosítson amatőr zenekarok részére. A zenekaroknak legyen lehetőségük információkat elhelyezni a zenekarról, a zenekar tagjairól, zenéjük műfajáról, a zenekar által készített albumokról, és a zenekar által tervezett fellépésekről. Az oldal látogatóinak legyen lehetőségük a zenekarok böngészésére, a zenekarok albumainak megtekintésére, keresni a zenekarok, illetve az albumok között.

Az oldalnak lehetőséget kell biztosítania, hogy a zenekarok bemutatkozhassanak rajta keresztül. Legyen lehetőség a zenekar történetét bemutatni, röviden bemutatni a zenekar

tagjait, és ezekhez kapcsolódó fényképeket elhelyezni. A zenekar tudja megadni az eddig elkészített albumait, az ezekhez kapcsolodó információkat, mint például tracklista, borító, helyezhesse el az oldalon. Ezen túl a zenekar elérhetőségét is lehessen elhelyezni. Az oldal látogatóinak legyen lehetőségük a különböző zenekarokat bemutató lapokat böngészni, az adott zenekarok albumait megtekinteni. A látogatók legyenek képesek az albumok között keresni, azokat böngészni, és az adott albumot elkészítő zenekar információt megtekinteni.

Az oldal szervezői időről-időre koncerteket szerveznek, az oldalon megjelenő zenekaroknak.A koncertek helyszínét és időpontját a szervezők a rendszeren kívül szervezik, majd a lehetséges körülményeket elhelyezik az oldalon. Ezen kívül meghatároznak egy műfajt is, ezt is elhelyezik az előbbi információk mellett. Az adott műfajú zenét játszó zenekarok jelentkeznek a koncertre, majd egyeztetnek a szervezőkkel.

1.4 Diagramok

Az UML különböző diagramjai a modellezett alkalmazást különböző nézőpontokból, különböző nézetekből szemléltetik. Ebből fakadóan az egyes diagramok között előfordulhatnak átfedések, azaz megtörténhet, hogy a rendszer ugyanazon pontját modellezzük rajtuk, de a diagramok típusától függően ugyanazt a dolgot más megvilágításban láthatjuk.

A strukturális diagramok a rendszer statikus jellemzőit, a statikus elemeket és ezek kapcsolatait, ezáltal a rendszer belső struktúráját modellezik. A strukturális diagramokon modellezett elemek egy keretrendszert definiálnak, amely keretrendszeren belül megvalósulhat az alkalmazás dinamikája, viselkedése. A strukturális diagramok:

- osztálydiagram
- objektumdiagram
- komponensdiagram
- alkalmazási diagram

A viselkedési diagramokon a strukturális diagramokon modellezett elemek dinamikáját, a rendszer működése közben megvalósított viselkedését modellezhetjük. Ezek a diagramok fejezik ki, hogy az erőforrások hogyan hajtják végre a feladataikat, hogyan működnek együtt a rendszer céljainak megvalósítása közben. A viselkedési diagramok:

- használati eset diagram
- aktivitás-diagram
- szekvencia-diagram
- kommunikációs diagram
- időzítési diagram

Ezen túl az UML lehetővé teszi a modell menedzsmentjével kapcsolatos diagramok használatát is. Segítségükkel például a rendszer és alrendszereinek struktúráját modellezhetjük. Ilyen diagramok:

- csomagdiagram

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 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. Például egy attribútumra vonatkozó megszorításnak fenn kell állnia az attribútumot tartalmazó objektum teljes életciklusán keresztül. Megszorításokat a modellelemek mögött kapcsos zárójelek között adhatjuk meg.

A kulcsszavas értékek segítségével a modellelemek specifikációját név-érték párok segítségével explicit módon további speciális jellemzőkkel egészíthetjük ki. Kulcsszavas értékeket a modellelem mellet név = érték formában, kapcsos zárójelek között adhatunk meg.

1.6 Kommentek

Természetesen lehetetlenség az összes, az alkalmazásfejlesztés során felmerülő fogalomra megfelelő 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.

2 Strukturális diagramok

2.1 Osztálydiagram

Az osztálydiagramok alapját az osztály fogalma jelenti. Az osztály az objektum-orientált paradigma központi fogalma, a valós világ fogalmainak magas szintű absztrakciója, amely lehetővé teszi az adatmodell és a funkcionális modell együttes kezelését. Az absztrakció során felmerülő adatokat attribútumokkal, a különböző viselkedéseket pedig metódusokkal modellezzük. Ennek megfelelően az osztályok modellezésekor annak nevét, attribútumait, és műveleteit kell meghatároznunk. Az UML az osztályok modellezésekor nem foglalkozik a metódusok implementációjával, csak az interfésszel, amelyen keresztül a metódusokhoz hozzáférhetünk, és az UML ezeket az interfészeket nevezi műveletnek. Az osztályok jelölése a diagramon egy téglalappal történik, amely három részre oszlik: névrész, attribútumrész, és műveletrész. Ezek a részek a nevüknek megfelelően az osztály definíciójának különböző információit tartalmazzák. A diagramon, annak részletezettségétől függően, az attribútum- és műveletrészek elmaradhatnak, egyedül a névrésznek kötelező szerepelnie.

Az osztály jelölésében a legfelső rész a névrész, amely az adott osztály nevét tartalmazza. Az elnevezés nagyon fontos, hiszen a modellezett rendszer alapvető erőforrásait, elemeit azonosítjuk vele, így a névnek egyedien és félreérthetetlenül kell azonosítania az osztály által definiált objektum típusát.

A névrész alatt helyezkedik el az attribútumrész, mely az osztály attribútumainak felsorolását, az attribútumok listáját tartalmazza. Az attribútumok definíciójánál az attribútumokra vonatkozóan például a következő információkat rögzíthetjük:

- láthatóság
- származtatás
- név
- típus
- multiplicitás
- alapértelmezett érték

Az attribútumok definíciójának alakja: [láthatóság] [/] név [: típus] [multiplicitás] [alapérték].

A láthatóság az objektum-orientált paradigma bezárás fogalmát hivatott megvalósítani, azt fejezi ki, hogy az adott attribútum az alkalmazás mely pontjáról érhető el, hivatkozható. Az UML a következő láthatósági szintek használatát teszi lehetővé:

- privát láthatóság, ekkor, az attribútum csak az adott osztályban használható, jelölése: -
- csomag láthatóság, ekkor az attribútum az osztályt tartalmazó csomagból hivatkozható, jelölése: ~
- publikus láthatóság, ekkor az attribútumot látja az összes osztály, jelölése: +
- védett láthatóság, ekkor az attribútumokhoz csak a leszármazott osztályok férhetnek hozzá, jelölése: #.

Az egyes szintek értelmezése megegyezik az objektum-orientált paradigma láthatósági szintjeinek értelmezésével.

Az attribútumok értéket kaphatnak valamilyen más attribútumból vagy adatból egy képlet alapján számítva. Ekkor azt mondjuk, hogy az attribútum értéke származtatott érték és ezt a neve előtt elhelyezett / jellel jelöljük.

Az attribútum neve kifejezi azt a szempontot, amely alapján az adott attribútum leírja az osztály példányait, ezért fontos, hogy lehetőleg minél kifejezőbb legyen, és az osztályon belül egyedinek kell lennie. Az attribútumok nevét kötelező megadni.

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.

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. Ha az attribútumhoz tartozó értékek száma egy konkrét szám, akkor a tartomány alsó és felső határát is ez a szám jelöli. Azt, hogy a tartomány felülről nem korlátos, a felső határ helyén egy *-gal jelölhetjük. Ha egy attribútum számossága nagyobb mint egy, értelmezhetünk az értékek között rendezettséget, és ezt a multiplicitást követően elhelyezett {isOrdered} tag-el fejezhetjük ki.

Az alapértelmezett érték megadásával azt az értéket határozhatjuk meg, amely értéket az attribútumok a példányok létrejöttekor felvesznek.

A műveletrészben a műveleteket, az osztály objektumainak lehetséges viselkedésmódjait soroljuk fel listaszerűen. A viselkedésmódok implementációjáról az osztálydiagram semmilyen információt nem tartalmaz, csupán az osztály objektumai által kínált szolgáltatások, viselkedések meghívásának feltételeit modellezi. A műveletek felsorolásakor a következő információkat rögzíthetjük:

- láthatóság
- név
- paraméterlista
- visszatérési érték típusa

A műveletek definíciójának alakja: [láthatóság] név [(paraméterlista)] : visszatérési típus.

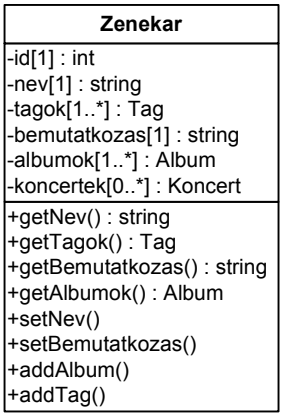
A láthatóság az attribútumok láthatóságához hasonlóan azt jelzi, hogy az adott művelet az alkalmazás mely pontjáról hívhatóak, aktiválhatóak. A megadható láthatósági szintek és jelölésük megegyezik az attribútumoknál elmondottakkal.

A név azonosítja az osztály objektumának egy viselkedésmódját, egy szolgáltatását, ennek megfelelően kifejezőnek kell lennie és kötelező megadni. A műveletek esetén az egyediség nem követelmény a név esetében, viszont a művelet szignatúrájának, azaz a név, paraméterlista, visszatérési típus hármasnak egy osztályon belül minden műveletre vonatkozóan egyedinek kell lenni.

A paraméterlista segítségével megadhatjuk a művelet mögött álló viselkedés megvalósításához szükséges inputot. Az egyes paramétereket név : típus alakban, egymástól vesszővel elválasztva adhatjuk meg.

A visszatérési érték típusa a művelet mögött álló viselkedés outputjának típusát adja meg.

A diagramon a példányszintű és osztályszintű műveletek között az utóbbiak definíciójának aláhúzásával tehetünk különbséget.



A különböző szoftverrendszerek erőforrások sokaságából épülnek fel, mely erőforrások együttműködnek egymással. Ahhoz, hogy az erőforrások egymással együtt tudjanak működni, kommunikálniuk kell, tehát tudatában kell lenniük a többi erőforrás létezésének, és ezen túlmenően valamilyen viszonyban, kapcsolatban is kell állniuk egymással. Az UML az erőforrások mint modellelemek kapcsolatainak modellezésére háromféle kapcsolattípus használatát ajánlja: asszociáció, általánosítás és függőség.

A rendszerben együttműködő objektumoknak a feladatok, felelősségek megosztásához szükségük van arra, hogy a rendszerben levő többi objektumot elérhessék, kommunikálhassanak egymással. Ezeket a kommunikációs útvonalakat, melyeken keresztül majd az objektumok információkat továbbíthatnak, az osztálydiagramokon asszociációkkal modellezük. Asszociációnak nevezzük az osztályok objektumai között lehetséges strukturális kapcsolatokat, azaz olyan kapcsolatok, amelyeket az objektum maga szolgáltat, így amikor az adott objektum elérhető, akkor kapcsolatai is elérhetőek. Az UML az asszociációt a résztvevő osztályok között húzott vonallal modellezi.

A leggyakoribbak a bináris asszociációk, ezek olyan asszociációk, amelyekben két osztály vesz részt. Természetesen előfordulhatnak magasabbrendű asszociációk is, ám ezeket többnyire felbontják, visszavezetik bináris asszociációk sorozatára.

Az asszociációk elnevezése ugyanolyan fontos, mint például az, hogy az osztályokat és az osztályok attribútumait elnevezzük. Az elnevezés elsődleges célja, hogy világosan, érthetően kifejezze az asszociáció célját. Ezen túl, az elnevezés fontos lehet abban az esetben, amikor két osztály között több asszociációt adunk meg, ekkor a kapcsolatok azonosításában is segíthet.

Az UML az asszociációkat jelölő vonalak végeit külön entitásként kezeli, amelyekhez az asszociációt leíró információkat rendelhetünk. Ezekkel az információkkal tulajdonképpen szabályokat definiálunk, amely szabályok az adott osztályok példányai közötti kapcsolatokra vonatkoznak. Az asszociációk végeinek legfontosabb jellemzői:

- szerepek
- számosság
- rendezettség
- navigálhatóság
- változtathatóság

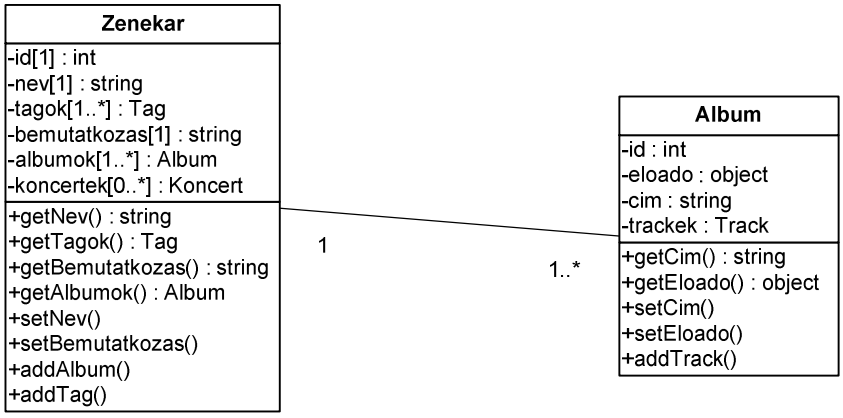
Az asszociáció végeihez szerepeket rendelhetünk, melyek a megfelelő osztály példányainak a kapcsolatban betöltött szerepét fejezik ki.

Az szerepek számossága azt jelzi, hogy az adott asszociációt megvalósító kapcsolatokban hány objektum vehet részt az egyik, illetve a másik oldalról. Az multiplicitás jelölése az attribútumok számosságának jelöléséhez hasonló, csak ebben az esetben nem kellenek a szögletes zárójelek. Az asszociáció egyik végéhez rendelt számosság azt fejezi ki, hogy az asszociáció másik oldalán álló osztály egy objektumához hány objektum kapcsolódhat az előbbi oldalról.

Abban az esetben, ha asszociáció egyik végének számossága nagyobb mint egy, az UML lehetővé teszi, hogy szükség esetén jelöljük az objektumok rendezettségét. Ezt az {ordered} tag elhelyezésével jelölhetjük.

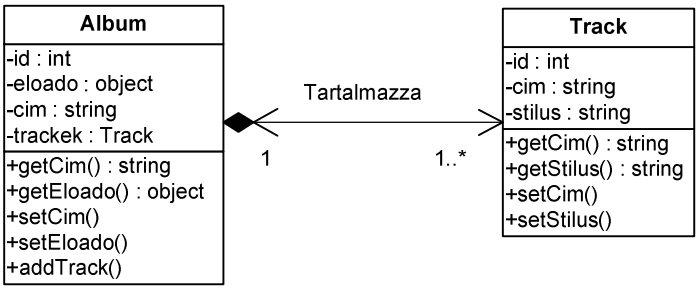
A navigálhatóság annak igényét fejezi ki, hogy az asszociációban résztvevő osztályok példányai a kapcsolaton keresztülnavigálva elérhetik a kapcsolódó objektumokat. Vagyis egy objektum elérheti a hozzá kapcsolódó objektumokat, ha az asszociációban feltüntettük a navigálhatóságot a megfelelő irányban. A navigálhatóság kifejezésére az asszociációt jelölő vonal végére helyezett nyílfejeket használhatjuk.

A szerepek változtathatósága az asszociációk által definiált kapcsolatokra értelmezhető műveleteket fejezi ki. A {frozen} tag elhelyezésével például előírhatjuk, hogy miután a kapcsolat létrejött, ne lehessen azt megváltoztatni, vagy törölni. Ha azt akarjuk kifejezni, hogy csak újabb kapcsolatok kialakítását szeretnénk engedélyezni, azt az {addOnly} tag segítségével tehetjük meg.



Az asszociációk nagy részében a résztvevő osztályok egyenrangúak, azaz egyik osztály sem alárendeltje a másiknak, és egymástól függetlenek, csak kommunikálnak, információt cserélnek egymással. Az aggregáció az asszociáció egy speciális típusa, amely az összetett, komplex elemek modellezését segíti, egy speciális asszociáció, mely két elem között fennálló egész/rész viszonyt fejez ki. Az aggregációban az „egész” állapotának része a „rész” állapota, de ugyanakkor a „rész” más „egészeknek” is a részét képezheti, és az „egész” irányítja a „részek” működését. Az UML az aggregációs viszonyt az asszociáció „egész” részénél elhelyezett rombusszal jelöli.

A kompozíció az aggregációnál is szigorúbb kapcsolattípus, amelyben az aggregációnál elmondottakon túlmenően az „egész” felelőssége a rész élettartamának menedzselése is. Azaz az „egész” határozza meg a „rész” létrejöttének és elhalálozásának körülményeit is, a „részek” élettartama is az „egésztől” függ, a „részek” nem létezhetnek „egészek” nélkül, és a rész kizárólag egyetlen egészhez tartozhat. A kompozíció jelölése egy olyan vonallal történik, amely az egész felől egy telt rombuszban végződik. Az UML a kompozíciós viszonyt az „egész” résznél elhelyezett telt rombusszal jelöli.



Gyakran szükség lehet arra, hogy az asszociációkról egyéb információkat is kifejezzünk, ne csak magát a viszony leírását modellezzük. Mivel az objektum-orientált paradigmában az információkat attribútumként modellezzük, ebből következik, hogy asszociációkat osztályokkal, úgynevezett asszociációs osztályokkal modellezzük. Az asszociációs osztályok, mivel osztályok, saját maguk is részt vehetnek asszociációkban.

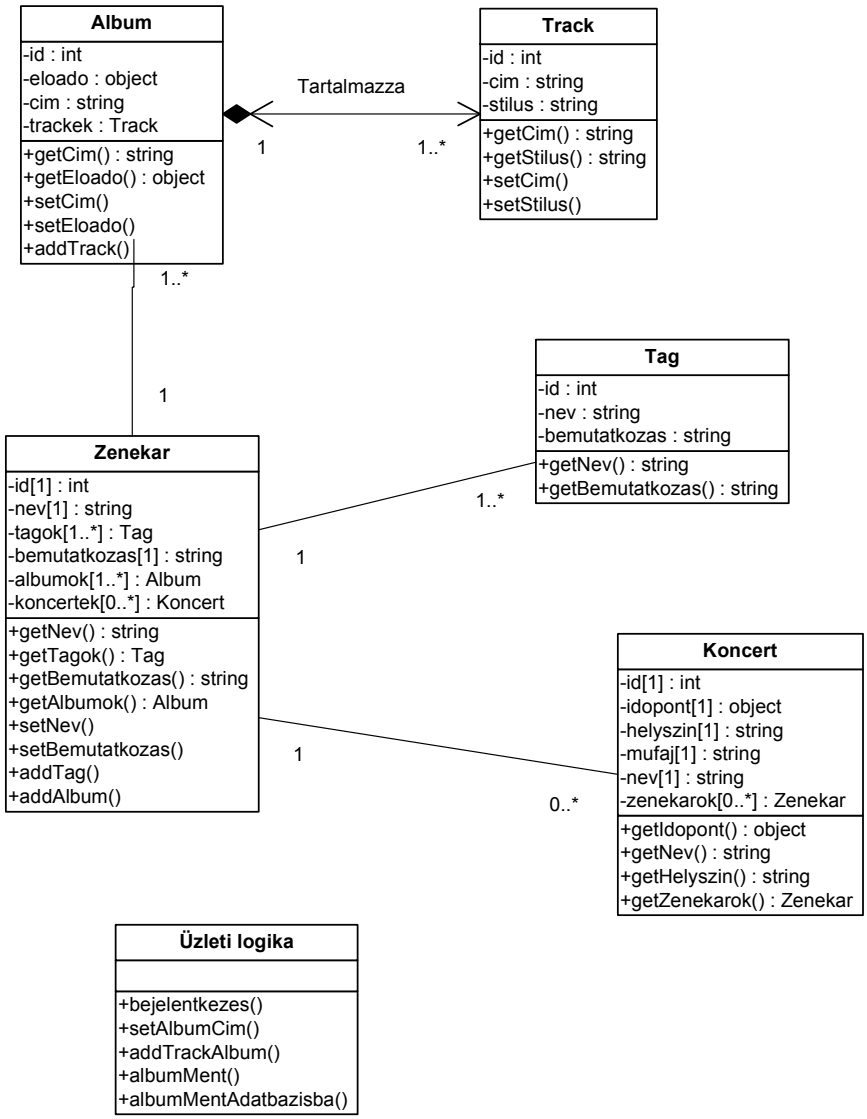
Az osztályok jellegzetességeinek összegyűjtése során észrevehetjük, hogy bizonyos jellemzők, viselkedések több osztályban is felmerülnek, ismétlődnek. Ha találunk olyan osztályokat, amelyek közös attribútumokkal, műveletekkel és asszociációkkal rendelkeznek, akkor lehetséges, hogy az osztályok által 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. Az UML az általánosítás viszonyt az alosztályból az ősosztályba vezető háromszögben végződő nyíllal jelöli. Az általánosítás viszonyhoz sztereotípiát és megszorítást is hozzárendelhetünk.

Az UML által támogatott harmadik kapcsolattípus a függőség. Az UML definíciója szerint egy modellelemtől függ egy másik modellelem, ha a definíciójának megváltozása a másik, azaz a függő elem megváltozását is eredményezheti. A függőséget nemcsak osztályok között értelmezhetünk, hanem többféle modellelem között, például csomagok között is. A függőségi viszonyok minősítéséhez sztereotípiákat használhatunk. Az UML a következő előredefiniált sztereotípiák használatát támogatja a függőségek minősítésére:

- <<use>>: a függő elem definíciója felhasználja a másik modellelem definícióját. Ez az alapértelmezett függőség, ezért csak akkor adjuk meg, ha hangsúlyozni szeretnénk, hogy nem más jellegű függőségről van szó
- <<derive>>: a függő elem származtatott, azaz csak koncepcionális, és csak a másik elemből számítható
- <<friend>>: olyan speciális használati módot jelöl, amikor a függő elem jogosult elérni a használt elem védett és privát jellegzetességeit is
- <<call>>: a függő elem meghívja a használt elemet vagy annak egy műveletét

- <<instantiate>>: a függő osztály a használt osztálynak megfelelő típusú objektumokat hoz létre
- <<instanceof>>: a függő elem a használt elem példánya
- <<refine>>: ezzel jelölhetjük, hogy a függő elem azonos a használt elemmel, de annak több részletét tartalmazza
- <<trace>>: nyomkövetéssel a fejlesztés során a modell kialakításának lépéseit ábrázoljuk

A függőséget a diagramon a függő elemtől, osztálytól a használt modellelemhez, osztályhoz húzott szaggatott vonalú sztereotípiával ellátott nyíllal jelölhetjük.


Album
-id : int
-eloado : object
-cim : string
-trackek : Track
+getCim() : string
+getEloado() : object
+setCim()
+setEloado()
+addTrack()
Tartalmazza
1
1..*
Track
-id : int
-cim : string
-stilus : string
+getCim() : string
+getStilus() : string
+setCim()
+setStilus()
1..*
1
Tag
-id : int
-nev : string
-bemutatkozas : string
+getNev() : string
+getBemutatkozas() : string
Zenekar
-id[1] : int
-nev[1] : string
-tagok[1..*] : Tag
-bemutatkozas[1] : string
-albumok[1..*] : Album
-koncertek[0..*] : Koncert
+getNev() : string
+getTagok() : Tag
+getBemutatkozas() : string
+getAlbumok() : Album
+setNev()
+setBemutatkozas()
+addTag()
+addAlbum()
1..*
1
1
0..*
Koncert
-id[1] : int
-idopont[1] : object
-helyszin[1] : string
-mufaj[1] : string
-nev[1] : string
-zenekarok[0..*] : Zenekar
+getIdopont() : object
+getNev() : string
+getHelyszin() : string
+getZenekarok() : Zenekar
Üzleti logika
+bejelentkezes()
+setAlbumCim()
+addTrackAlbum()
+albumMent()
+albumMentAdatbazisba()

2.2 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 objektum jelölése egy téglalappal történik, melybe aláhúzva beleírjuk az adott objektum elnevezését. Az objektum neve két részből áll: az objektum azonosító nevéből és az objektum osztályának nevéből, melyeket kettősponttal választunk el egymástól. Anonim objektumok esetén, vagyis olyan szituációkban, amikor azt akarjuk kifejezni, hogy egy osztály minden példányára érvényes a modellezett körülmény, az elnevezésből csak a kettőspont és utána az osztály neve szerepel.

Az osztályok attribútumokkal rendelkeznek, és az osztály példányai ezen attribútumokra értékeket vesznek fel. Az attribútumértékek együttesen meghatározzák az adott objektum állapotát. Egy objektum által felvett attribútumértékeket az objektum névrésze alatt, az osztálydiagramhoz hasonlóan attribútumrésznek nevezett szekcióban sorolhatjuk fel. Ennek alakja: az attribútum neve után megadjuk a konkrét értéket. Nem kötelező az összes attribútumértéket feltüntetni, elég azokat megadni, amelyek szükségesek a helyzet jellemzése szempontjából.

Az objektumok az attribútumaikon túl, kapcsolatokkal is rendelkeznek. Azt, hogy az egyes objektumok milyen kapcsolatokkal rendelkezhetnek, az osztálydiagramon az asszociációk megadásával rögzítettük. Az objektumok között fennálló kapcsolatok az osztályok között modellezett asszociációk példányai, előfordulásai. Az objektumok között fennálló kapcsolatokat hasonlóan az osztályok közötti asszociációkhoz az objektumok között húzott vonallal jelölhetjük.

2.3 Komponensdiagram

Az alkalmazás logikai tervének elkészülte után a terv fizikai implementációjának megtervezése lehet a következő lépés. Ennek az első és az egyik legfontosabb kérdése az alkalmazás fizikai szerkezetének modellezése. A komponensdiagram célja az alkalmazást

alkotó fizikai szoftvermodulok és azok egymáshoz való viszonyának modellezése. A diagram alkalmas a kész alkalmazás fizikai szerkezetének vázolására, valamint a fejlesztés során használt állományok közötti viszonyok szemléltetésére.

A komponens egy olyan eszköz, amelynek segítségével az alkalmazás különböző fejlesztés alatt álló, vagy már kész alkotóelemeit összefoghatjuk. Komponensek lehetnek például:

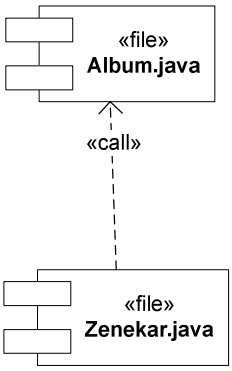
- forrás-állományok
- kód-állományok
- programkönyvtárak
- futtatható állományok
- dokumentumok
- adatállományok

A komponenseket a diagramon egy téglalappal jelölhetjük, amelynek baloldalán két téglalap alakú címkét veszünk fel.

A komponenseket névvel láthatjuk el, amelyet a téglalapba írunk, és szintén a téglalapba írva sztereotípiával kifejezhetjük azt a szerepet, amelyet az adott komponens az alkalmazás architektúrájában betölt. Az UML által előredefiniált, komponensekhez rendelhető sztereotípiák:

- <<executable>>: jelöli a futtatható programot tartalmazó állományt
- <<library>>: programkönyvtár, melyre egy program hivatkozik, a hivatkozás lehet statikus vagy dinamikus
- <<table>>: adatbázistábla
- <<file>>: forráskódot, vagy adatot tartalmazó állomány
- <<document>>: jelöli a dokumentumot tartalmazó állományt

Azokat a komponenseket, amelyek végrehajtható kódot, függvényeket, programokat tartalmaznak, a funkcióikon keresztül érhetjük el. Ezeket a funkciókat kiemelhetjük, csoportosíthatjuk interfészekbe, és a diagramon az interfészek szokásos jelölésével modellezhetjük. Az ily módon kiemelt interfészeket a komponens elemei implementálják, realizálják.



2.4 Alkalmazási diagram

Az alkalmazási diagram segítségével az alkalmazást működtető hardver egységeit és az azok között fennálló fizikai kapcsolatokat modellezhetjük, azaz a végrehajtási környezetet, architektúrát szemléltethetjük. Ezen felül az alkalmazást alkotó egyes komponensek elhelyezkedését is modellezhetjük. A diagrammal jól ábrázolható az alkalmazás készülékigénye, valamint például a több gépen futó kliens/szerver alkalmazások fizikai felépítése és az azokon működő komponensek.

A diagram alapvető alkotóeleme a csomópont, amely egy olyan eszközt reprezentál, amely munkát végez. A csomópont többnyire egy számítógépes egységet, egy hardverelemet jelképez, mint például egy lemezmeghajtó, vagy egy szervergép. A csomópontokat az azonosítás céljából névvel kell ellátni. A csomópontokat a diagramon egy kockával jelölhetjük. A csomópontok lényegi tartalmaként ábrázolhatóak a rajta elhelyezkedő komponensek, a lényegesebb adatelemek pedig objektumokként jeleníthetőek meg. A csomópontokhoz attribútumokat és műveleteket is definiálhatunk, és ezen felül a csomópontok kapcsolatban is állhatnak más csomópontokkal, azaz asszociációkat is értelmezhetünk közöttük, amelyeket a csomópontokat összekötő vonalak segítségével jelölhetünk.

3 Viselkedési diagramok

3.1 Használati eset diagram

A használati eset diagram egyedi az UML diagramok között abban a tekintetben, hogy a rendszer felhasználóinak a rendszerrel szemben támasztott elvárásait, követelményeit modellezi, leírja azokat a jellemzőket, amelyeket a felhasználók a rendszertől elvárnak. Ugyanakkor a részletekről, azaz a diagram segítségével összegyűjtött követelmények megvalósításáról, implementációjáról semmiféle információt nem tartalmaz a diagram.

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

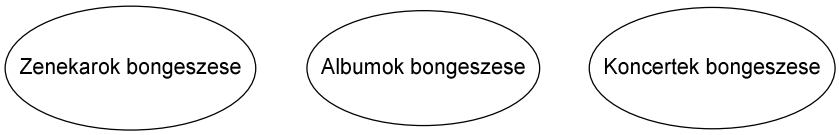
- a rendszeren kívül eső, de ahhoz kapcsolódó szereplők, aktorok összegyűjtése
- a rendszer viselkedését, funkcióit, céljait kifejező használati esetek összegyűjtése
- az előző lépésekben összegyűjtött elemek kapcsolatainak vizsgálatával a modell finomítása

A használati eset diagram célja egy olyan rendszeren kívüli nézőpont biztosítása, amely a rendszer és a rendszeren kívül eső, de a rendszerhez kapcsolódó dolgok viszonyát jeleníti meg.

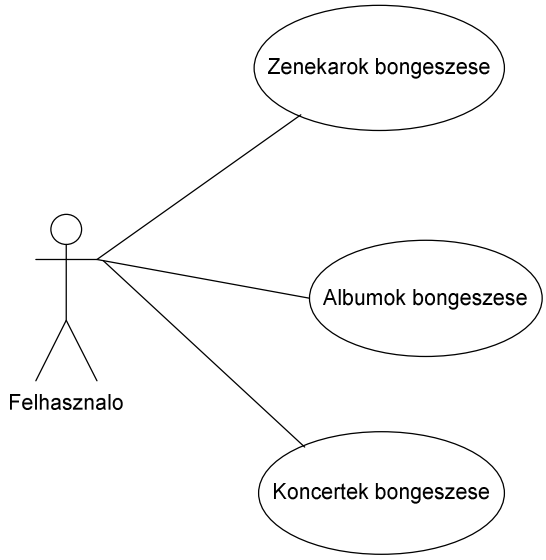
A követelmények feltárásának első lépése annak meghatározása, hogy a rendszeren kívül eső elemek milyen módon kapcsolódnak a rendszerhez. Ezeket a kapcsolódásai módokat nevezi az UML aktoroknak. Az aktor tulajdonképpen egy szerepkört fejez ki, azaz a rendszerhez kapcsolódó elemek egy típusát azonosítja. Aktor lehet egy felhasználó, egy hardvereszköz, vagy akár egy másik alkalmazás is. Természetesen előfordulhat, hogy például egy személy több szerepkörben is kapcsolódik a rendszerhez, és ennek a fordítottja is igaz lehet, azaz hogy több személy is ugyanabban a szerepkörben kapcsolódik a rendszerhez. Az aktorok jelölésére többféle lehetőség kínál az UML. Azokat az aktorokat, amelyek mögött személyek állnak, általában egy pálcikaemberrel jelöljük, és alá írjuk az adott aktor

elnevezését. Másik lehetőség a jelölésre egy <<actor>> sztereotípiájú téglalap, amelybe beleírjuk az aktor elnevezését.

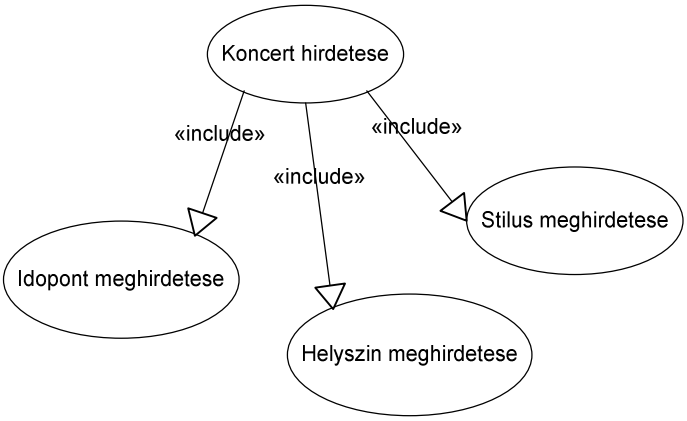
Az előző lépésben összegyűjtött aktorok különböző célokkal kapcsolódnak a rendszerhez, különböző eredményeket várnak a rendszertől, melyeket a rendszer funkciók formájában tesz elérhetővé. A következő lépés a követelmények elemzésében annak meghatározása, hogy az aktorok mely funkciókon keresztül kapcsolódnak a rendszerhez. Ezeket a kapcsolódási pontokat nevezi az UML használati esetnek. A használati eset a rendszer egy olyan fontos viselkedésmódját azonosítja, amely során az aktor számára valamilyen eredmény vagy érték keletkezik, és amelynek hiányában a rendszer nem használható a céljának megfelelően, azaz a rendszer nem teljesíti a vele szemben támasztott követelményeket. A használati eseteket a diagramon egy oválissal jelöljük, amelybe beleírjuk az adott használati eset elnevezését.



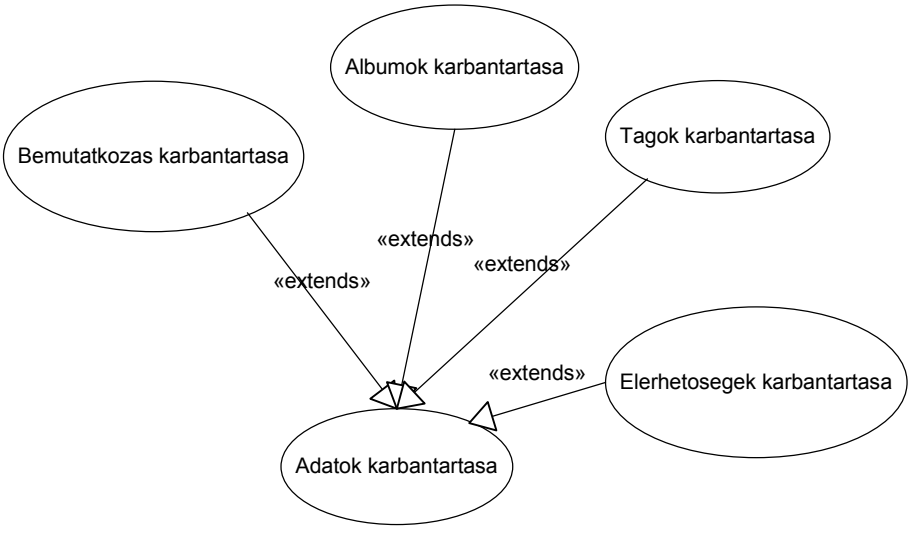
A use case diagrammal kapcsolatban használt asszociáció fogalma eltér asszociáció osztálydiagramnál vett jelentésétől, azt fejezi ki, hogy egy aktor kommunikál egy használati esettel. Ez az egyetlen olyan kapcsolat, amely aktor és use case között fennállhat. Jelölése egy az adott aktort és használati esetet összekötő vonallal történik.



Az <<include>> típusú kapcsolatot két irányból is megközelíthetőek, két különböző dolgot is kifejezhetünk a segítségével. A tervezett alkalmazás funkcióit kifejező használati esetek tervezésénél előfordulhat, hogy találunk egy olyan részfunkciót, amelyet valamilyen okból külön hangsúlyozni szeretnénk. Ebben az esetben a részfunkciót <<include>> típusú kapcsolattal csatoljuk az őt tartalmazó funkcióhoz. Előfordulhat az is, hogy a használati esetek tervezésénél észrevesszük, hogy a rendszernek több különböző ponton is ugyanazt a viselkedést, ugyanazt a funkciót kell nyújtania az aktorok felé, esetleg ugyanezt a funkciót már rögzítettük használati esetként. Ilyenkor szintén <<include>> típusú kapcsolatot használhatunk, ezzel kifejezve az újrafelhasználás fogalmát. Az ilyen típusú kapcsolatok azt fejezik ki, hogy van egy alap használati eset, amely a rendszer egy funkcióját, viselkedését modellezi, és ezt a viselkedést kibővíti egy másik használati eset. Az alap használati eset mintegy meghívja a kiterjesztő használati esetet, és ez a hívás a viselkedés megvalósításában feltétel nélkül, mindig bekövetkezik. Azt, hogy ez a hívás mikor következik be, azt az alap használati eset határozza meg. Az <<include>> típusú kapcsolatokat az alap használati esettől a kiterjesztő használati esethez húzott <<include>> sztereotípiájú nyíllal jelölhetjük.

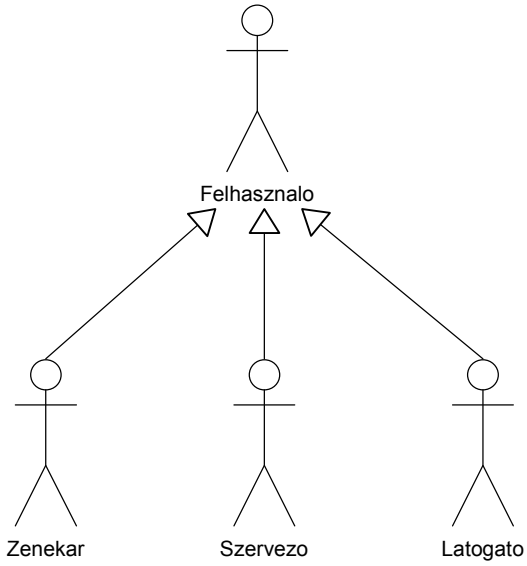


Az <<extend>> típusú kapcsolatoknál az előbbiekhez hasonlóan arról van szó, hogy van egy alap használati eset, és ennek a viselkedését kiterjeszti, kibővíti egy másik használati eset. Azonban ez a kapcsolat másképp működik: az alap használati eset kiterjesztése opcionális, a kiterjesztő használati eset úgymond beszúrja saját magát az alap használati esetbe félbeszakítva annak működését, mégpedig egy adott feltétel teljesülése esetén. Ehhez az alap használati esetben úgynevezett kiterjesztési pontokat kell definiálni, mely pontokon feltételek kiértékelése alapján dől el, hogy megtörténik-e az alap használati eset kiterjesztése. Az <<extend>> típusú kapcsolatokat a kiterjesztő használati esettől az alap használati esethez húzott <<extend>> sztereotípiájú nyíllal jelölünk. A kiterjesztési pontokat megadhatjuk az alap használati eset jelölésében, a használati eset nevétől egy vízszintes vonallal elválasztva.



Az általánosítás/pontosítás viszonyról mind aktorok, mind pedig használati esetek kapcsán beszélhetünk, és jelentése megegyezik az objektumorientált paradigma fogalmával. Miután összegyűjtöttük az aktorokat, célszerű megvizsgálni, hogy léteznek-e említésre méltó alváltozatai az egyes aktoroknak, illetve hogy egy használati eset végrehajtása során több

szereplő is betöltheti-e ugyanazt a szerepet. Ekkor a közös vonásokat kiemelhetjük egy közös ősbe, a leszármazott aktorok pedig ugyanazokat a használati eseteket kezdeményezhetik, mint az ősük, de ennél többet is tehetnek. Hasonlóan, a használati esetek esetén elvégezhetjük ezeket a vizsgálatokat: ha több használati esetnek hasonló a szerkezete, viselkedése és célja, akkor a közös részeket kiemelhetjük egy általános használati esetbe. A leszármazottak öröklik az ősük struktúráját, viselkedését és kapcsolatait, ezeket felülbírálhatják, és további viselkedéseket adhatnak az őstől örökölt viselkedésekhez. Az UML az általánosítás/pontosítás viszonyt aktorok és használati esetek esetében is a pontosított elemtől az általános elemhez húzott háromszögben végződő nyíllal jelöli.



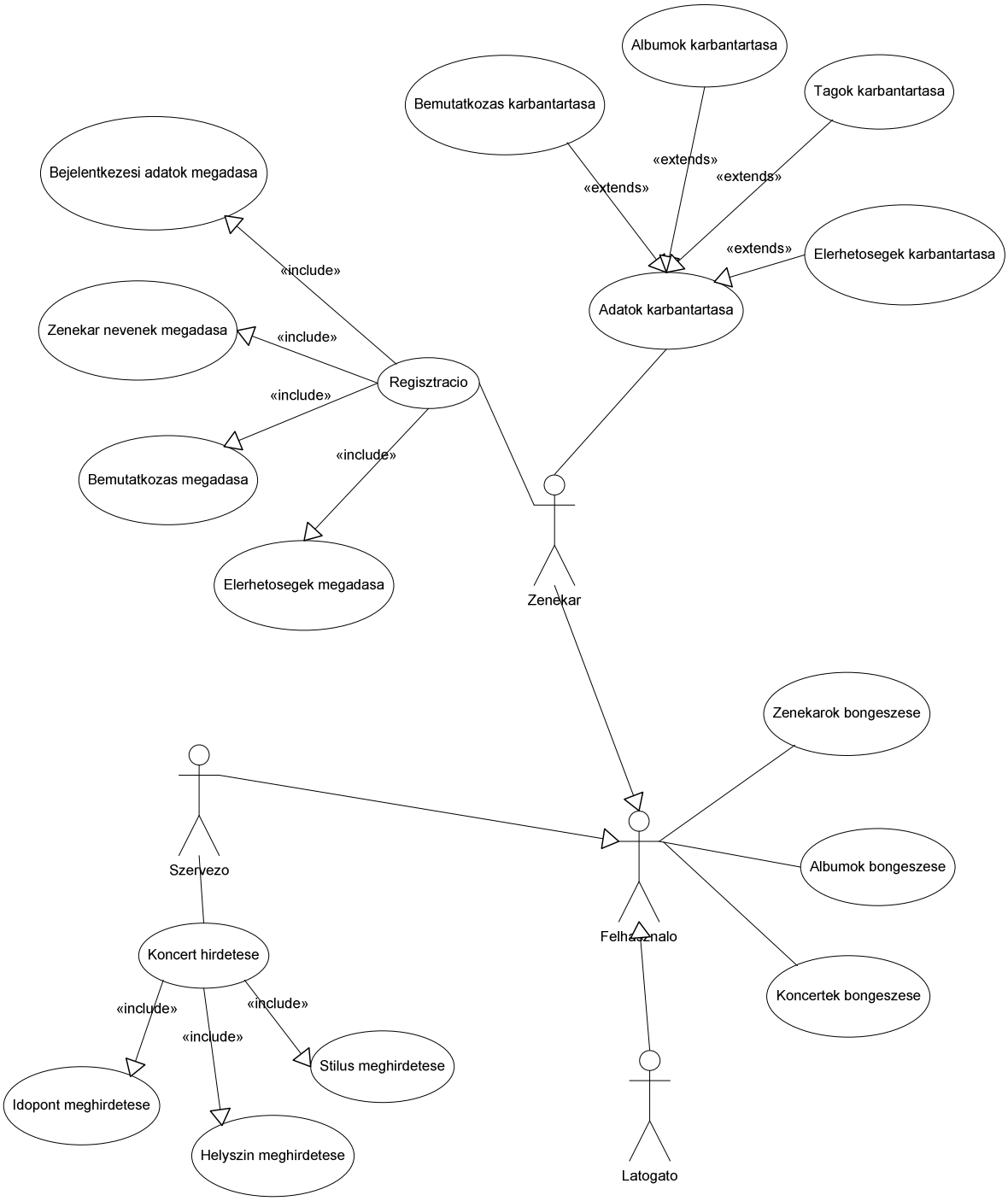
A használati eset diagram bár kiváló eszköz az aktorok és a rendszer jellemzői közötti kapcsolatok modellezésére, hiányoznak belőle azok a részletek, amelyek konkrétan a használati esetek megvalósítását jelentő viselkedéseket magyarázzák. Ezért általában a használati esetet szöveges dokumentumok magyarázzák, melyeket használati eset narratíváknak nevezünk. Egy használati eset narratívája általában a következő információkat tartalmazza:

- kezdeti feltételezések: ezek olyan információk, amely a rendszer azon állapotát írják le, amely szükséges ahhoz, hogy a használati esetben megadott viselkedést a rendszer meg tudja valósítani, ha ez a rendszerállapot nem teljesül, a rendszernek nem szabad a használati esetet elindítania
- előfeltételek: szintén a rendszer egy olyan állapotának a leírása, amelynek fenn kell állnia ahhoz, hogy a használati esettel modellezett viselkedés megvalósulhasson,

azonban ezen körülmények, feltételek teljesülését maga a használati eset vizsgálja, és adott esetben félbeszakítja saját működését

- triggerek: a használati esetnek működésének valahogy el kell kezdődnie, ez többféle esemény hatására bekövetkezhet. Ezeket a lehetséges eseményeket is rögzíteni kell, lehetnek például egy menüpont kiválasztása, egy eszköz jelzést küld a rendszernek, vagy a rendszeren belül valamilyen körülmény bekövetkezik
- lefutás: ez a rész lépésről lépésre leírja az adott használati eset lefutását, azaz aktorok és a használati esetek közötti interakciókat, ez a rész lehet az alapja az aktivitás-diagramok elkészítésének
- befejeződés: a használati eseteket általában egy körülmény indítja el, azonban a használati esetnek többféle végeredménye, kimenetele lehet, beleértve a helyes müködésből és a különböző lehetséges hibákból adódó befejeződéseket
- végfeltételek: a rendszer azon állapota, amelybe a használati eset befejeződését követően kerül

A használati eset diagramokhoz kapcsolódik még egy szöveges dokumentum, amely az egyes használati esetek forgatókönyvét írja le. A használati eset a rendszer egy elsődleges célját azonosítja, azonban ezen cél elérésére irányuló folyamatnak a különböző körülmények hatására általában többféle kimenetele lehet. A forgatókönyv egy használati eseten belül lehetséges útvonalat ír le, tulajdonképpen a használati eset egy példánya, a használati eset egy lehetséges megvalósulása.

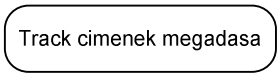

Albumok karbantartasa
Bemutatkozas karbantartasa
Tagok karbantartasa
«extends»
«extends»
«extends»
Bejelentkezesi adatok megadasa
«extends»
Elerhetosegek karbantartasa
«include»
Adatok karbantartasa
Zenekar nevenek megadasa
«include»
Regisztracio
«include»
«include»
Bemutatkozas megadasa
Elerhetosegek megadasa
Zenekar
Zenekarok bongeszese
Szervezo
Albumok bongeszese
Koncert hirdetese
Koncertek bongeszese
«include»
«include»
«include»
Stilus meghirdetese
Idopont meghirdetese
Helyszin meghirdetese
Latogato

3.2 Aktivitás-diagram

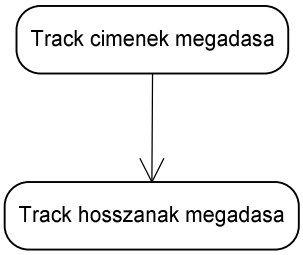
Az aktivitás-diagram az üzleti modellezésben használt folyamatábra UML-beli megfelelője. A diagram az alkalmazás dinamikáját, időben lezajló változását aktív oldalról, a végrehajtandó tevékenységek sorrendiségének meghatározásával ábrázolja, eszközeinek segítségével a különböző folyamatok vezérlései kiválóan modellezhetőek. Minden esetben, amikor valamilyen folyamatot akarunk modellezni aktivitás-diagramot használunk. Ezek a folyamatok lehetnek például:

- egy függvény algoritmusa
- egy használati eset működési módja, lehetséges lefutásai
- egy részrendszeren belüli használati esetek együttműködése
- a rendszert alkotó részrendszerek együttműködése.

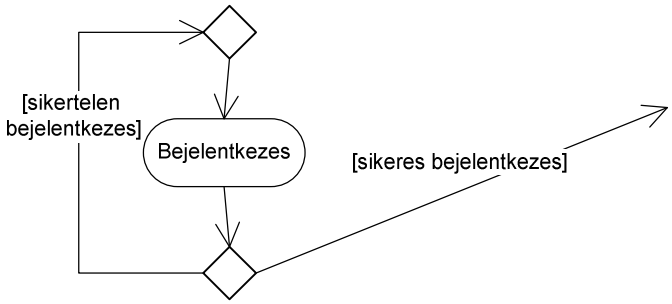
Az aktivitás a modellezett folyamat egy olyan lépését, állapotát jelenti, amikor valamilyen tevékenységet végre kell hajtani. Ez a tevékenység általában nem egy atomi műveletet jelent, a tevékenység a diagram részletezettségi szintjétől függően további altevékenységekre bontható, amelyeket adott esetben egy újabb aktivitás-diagramon modellezhetünk. Ennek megfelelően egy aktivitás jelenthet például egy metódushívást, vagy éppen egy egyszerű utasítás végrehajtását. Az aktivitást a diagramon ívelt oldalú téglalappal jelöljük, amelybe beleírjuk a tevékenység elnevezését.



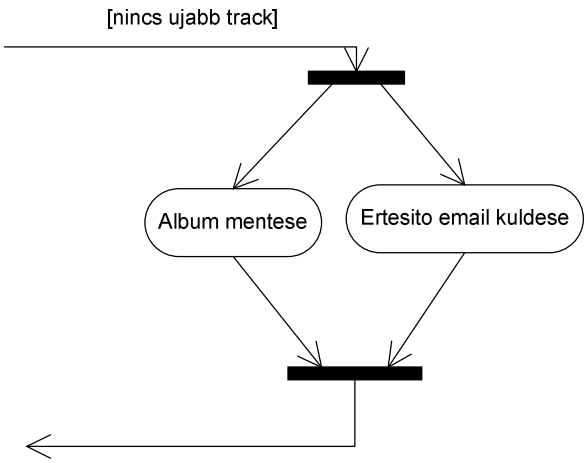
Az aktivitás-diagramon az aktivitásokat úgynevezett átmenetekkel kapcsoljuk össze. Az átmenet azt fejezi ki, hogy egy aktivitás végrehajtása befejeződött, és kezdődhet a következő tevékenység végrehajtása. Így az átmenetekkel tulajdonképpen az aktivitások között egy időbeli sorrendet határozunk meg. Az átmeneteket a diagramon a megelőző aktivitásból a rákövetkező aktivitásba vezető nyíl segítségével jelölhetjük.



A modellezett folyamat logikájában előfordulhat, hogy egy átmenet végrehajtását valamilyen feltételhez szeretnénk kötni. Ezt őrszemek segítségével tehetjük meg, ami egy az átmenetet jelölő nyíl mellett szögletes zárójelek között elhelyezett feltételt jelent. Ekkor az adott átmenet pontosan akkor következik be, ha a megadott feltétel igaz értékű. Ha egy feltétel alapján több alternatív aktivitás közül szeretnénk választani, akkor ezt egy rombusszal, és az abból kiinduló átmenetekkel modellezhetjük. Ekkor minden átmenethez egy őrszemet kell rendelnünk a feltétel értékei alapján, és ezeknek az ágaknak egymást kölcsönösen kizáróknak kell lenniük.

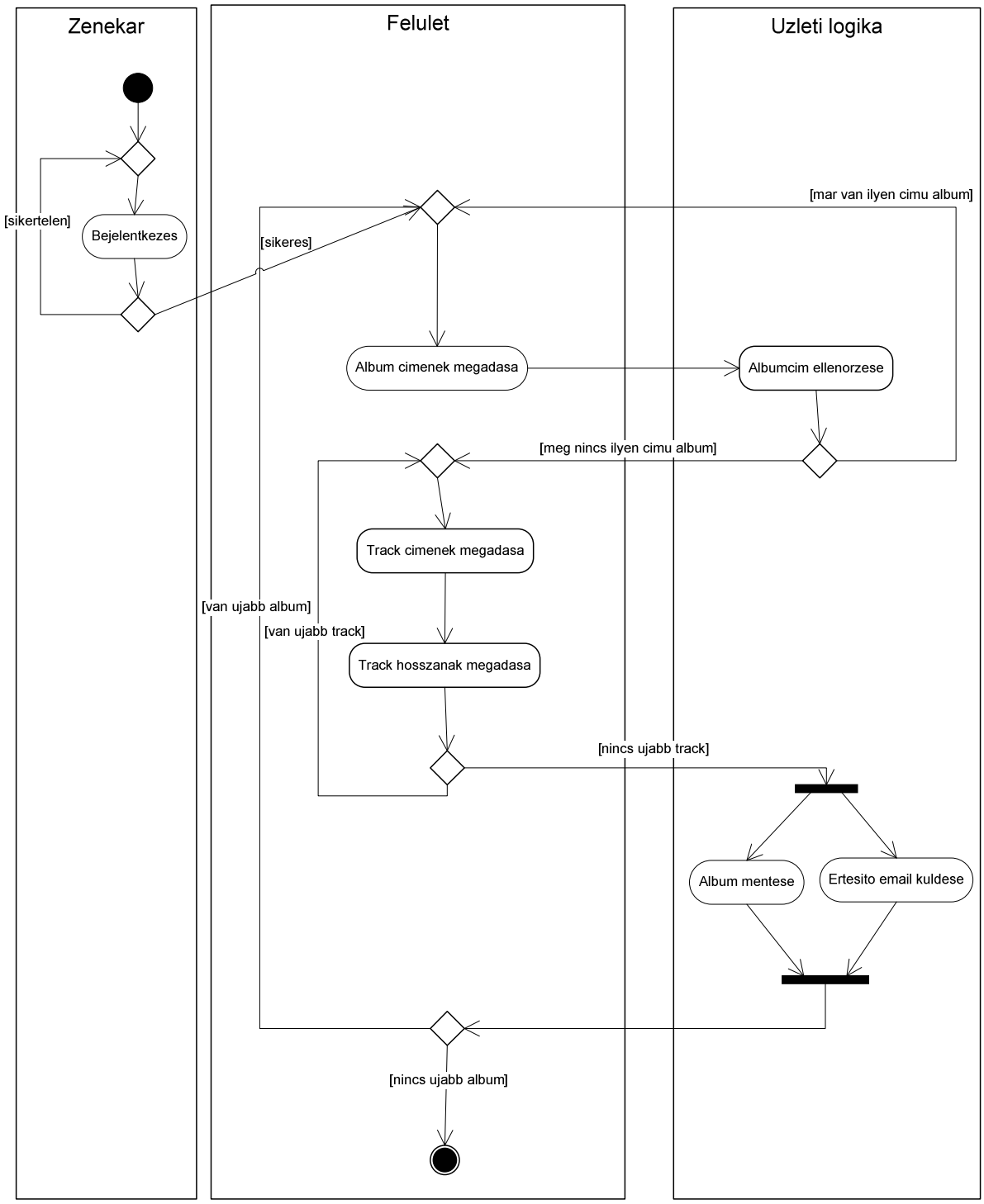


Az UML támogatja a párhuzamos végrehajtás jelölését, de fontos, hogy ezekkel az eszközökkel csupán azt hangsúlyozzuk, hogy a megadott tevékenységek párhuzamosan is végrehajthatóak, és nem jelenti azt, hogy az adott tevékenységeket kötelező így implementálni. Azt, hogy egy folyamatban több aktivitást párhuzamosan végre lehet hajtani, egy úgynevezett szinkronizációs vonal segítségével jelölhetjük. A szinkronizációs vonalba egy átmenet tart, és onnan több átmenet indul, ezzel jelölve, hogy a végrehajtás párhuzamosan, külön szálakon történik. A szinkronizációs pontokat, vagyis azokat a pontokat, amikor a párhuzamosan végrehajtott szálak újra egyesülnek, szintén egy szinkronizációs vonallal jelölhetjük, amelybe több átmenet tart, és egy átmenet indul ki. A szinkronizációs vonalat egy vastagított vonallal jelölhetjük.



Az UML a folyamat elkezdésének és befejeződésének jelölésére két pszeudó-állapotot javasol, ezek a kezdőállapot és végállapot. Kezdőállapotból egy diagramon egy szerepelhet, ebből csak kiindulhat átmenet, míg végállapotból több is szerepelhet a diagramon, hiszen egy folyamatnak tetszőleges számú kimenetele lehet, és a végállapotból nem indulhat ki átmenet. A kezdőállapotot egy körlappal, a végállapotot egy körben levő körlappal jelölhetjük.

Az aktivitás-diagram alaphelyzetben semmiféle információt nem tartalmaz arról, hogy az egyes tevékenységeket kinek a felelőssége végrehajtani. Ennek jelölésére feloszthatjuk a diagramot úgynevezett sávokra, majd a sávokhoz aktorokat, objektumokat rendelhetünk. Ezután a diagramot úgy kell kialakítani, hogy minden aktivitás abba a sávba kerüljenek, amely aktor, objektum felelőssége az adott aktivitás végrehajtása.


Zenekar
Felulet
Uzleti logika
[sikertelen]
Bejelentkezes
[sikeres]
[mar van ilyen cimu album]
Album cimenek megadasa
Albumcim ellenorzese
[meg nincs ilyen cimu album]
Track cimenek megadasa
[van ujabb album]
[van ujabb track]
Track hosszanak megadasa
[nincs ujabb track]
Album mentese
Ertesito email kuldese
[nincs ujabb album]

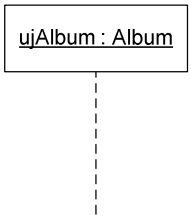
Interakció diagramok

Egy objektum-orientált rendszerben a rendszert alkotó objektumok nem egymástól függetlenül léteznek és működnek, hanem mint azt az osztálydiagramnál és objektumdiagramnál láttuk, egymással kapcsolatban állnak, és ezeken a kapcsolatokon keresztül kommunikálnak, egymást különböző feladatok elvégzésére kérik, azaz együttműködnek, interakciót folytatnak egymással. A strukturális diagramok, mint például az osztálydiagram vagy az objektumdiagram definiálják, leírják az objektumokat és az objektumok között fennálló kapcsolatokat, azonban semmiféle információt nem tartalmaznak arra vonatkozóan, hogy az objektumok hogyan kommunikálnak, viselkednek a rendszer funkcióinak megvalósítása közben.

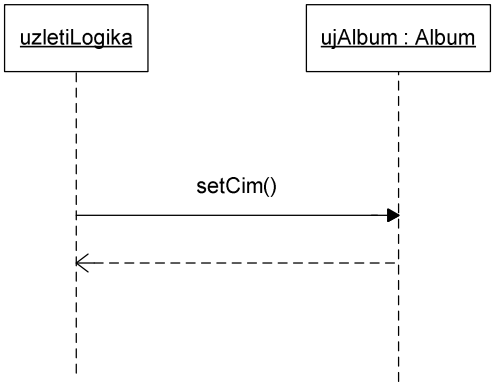
Az interakció diagramok az összetett feladatok megvalósítása során együttműködő objektumok együttes viselkedését szemléltetik. Az egyik interakció diagram a szekvencia-diagram, amellyel az interakciók időbeliségét, az üzenetek egymásutániságát hangsúlyozhatjuk. A másik interakció diagram a kommunikációs diagram, ezzel az objektumok szerveződése, felépítése mentén szemléltethetjük az interakciót folytató objektumokat és üzeneteiket. A harmadik ismertetett interakció diagram az időzítési diagram, ennek segítségével az interakciót folytató objektumok állapotának változását modellezhetjük az idő függvényében.

3.3 Szekvencia-diagram

A szekvencia-diagramon az interakcióban résztvevő objektumokat az objektumdiagramból vett jelöléssel, és egy abból kiinduló függőleges irányú szaggatott vonallal jelöljük. Ez a vonal az úgynevezett életvonal, amely az adott objektum interakcióbeli élettartamát szimbolizálja. Az objektumokat a diagram tetején, a diagram vízszintes tengelye mentén egymás után soroljuk el, az ezekből lefelé induló életvonalak pedig a diagram függőleges tengelye mentén az idő múlását szemléltetik.



Az objektumok közötti kommunikáció egyik alapvető formája az üzenetküldés. A kommunikáció megnyilvánulhat egy metódushívásban, egy jelzésben, vagy egy példány létrejöttét vagy elhalálozását kiváltó üzenetben. Az üzeneteket a küldő objektum életvonalától a fogadó objektum életvonaláig húzott nyíllal jelöljük. A különböző típusú üzeneteket a nyíl formájával különböztetjük meg. A szinkron üzenet egy olyan üzenet, amelyet a küldő objektum elküld a fogadó objektumnak, az feldolgozza a kérést, megválaszolja azt, majd a vezérlés visszakerül a küldő objektumhoz, a lényeg hogy a küldő objektum a visszatérésig nem kezdhet bele semmilyen újabb tevékenység végrehajtásába. Ezzel szemben aszinkron üzenetekről beszélünk, ha a küldő objektum feladata, felelőssége csupán az üzenet elküldése, amelyet aztán a fogadó objektum vagy megválaszol vagy nem, de a küldő objektumnak nem kell várnia a válaszra. A szinkron üzeneteket a küldő objektum életvonala és a fogadó objektum életvonala között húzott telt fejű nyíllal, az aszinkron üzeneteket a küldő objektum életvonala és a fogadó objektum életvonala között húzott félfejű nyíllal jelöljük. A szinkron üzeneteknél említett visszatérést, amely nem egy üzenetküldés, csupán annak jelölése, hogy az üzenet feldolgozása megtörtént, a fogadó objektum életvonalától a küldő objektum életvonalához húzott szaggatott vonalú egyszerű nyíllal jelölhetjük.

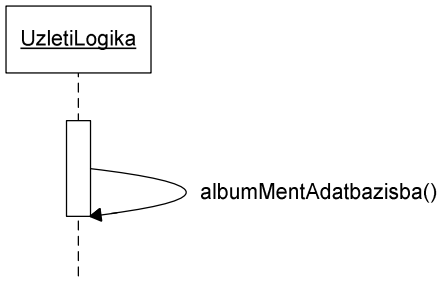


Az objektumok által egymásnak küldött üzenetek sorrendjét az üzeneteket jelölő nyilak egymáshoz viszonyított függőleges helyzete fejezi ki, a feljebb elhelyezkedő üzenetek hamarabb bekövetkeznek, mint a lejjebb elhelyezkedő üzenetek. Ugyanakkor az üzenetek közötti távolság semmilyen időmennyiséget nem fejez ki, csupán a nyilak egymáshoz viszonyított, relatív helyzete számít.

Az üzeneteket jelölő nyilak mellett magáról az üzenetről helyezhetünk el információkat. Egyrészt meg kell adni a küldött üzenet nevét, másrészt opcionálisan olyan egyéb információkat is megadhatunk, mint például az üzenet argumentumai, vagy ha olyan üzenetről van szó, amelynél beszélhetünk visszatérésről, akkor a visszatérési érték típusát.

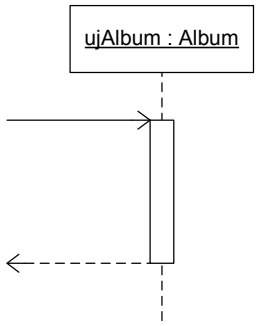
Feltételes üzenetküldést az üzenetet jelképező nyíl felett az üzenet neve előtt szögletes zárójelekben elhelyezett logikai kifejezés segítségével modellezhetünk. Iterációt, azaz annak tényét, hogy a fogadó objektumnak egy üzenet többször egymásután elküldésre kerül, szintén az üzenet neve előtt jelezhetjük, egy csillaggal illetve egy a végrehajtások számát kifejező értékkel.

Előfordulhat, hogy egy objektum valamelyik saját műveletét kell meghívja, ekkor öndelegációról beszélünk, az üzenetet jelző nyíl az adott objektum életvonalától indul, és oda is tér vissza.

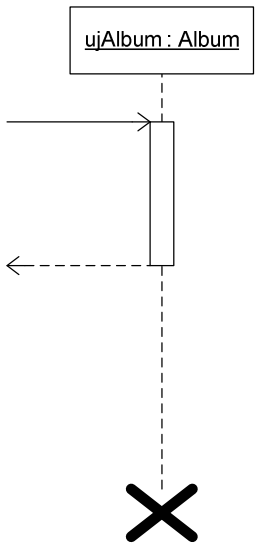


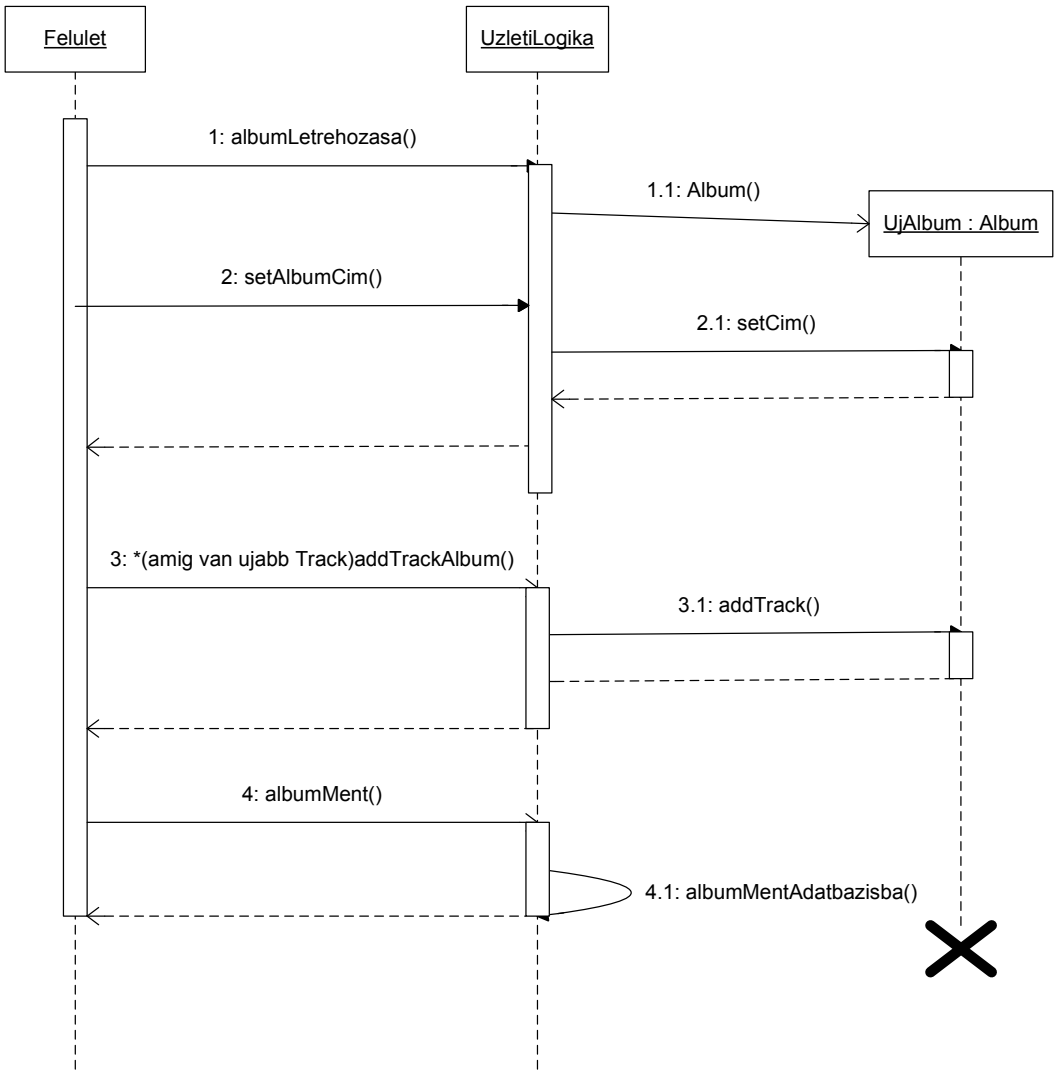
A szekvencia-diagram lehetővé teszi az objektumok aktivációjának illetve deaktivációjának jelölését is. Az aktiváció azt jelenti, hogy az objektum kapott egy üzenetet, és ez kiváltott valamilyen viselkedést az objektum részéről, azaz az objektum éppen valamilyen üzenetet dolgoz fel, valamilyen tevékenységet hajt végre. Ez a tevékenyég lehet valamilyen közvetlenül az objektum által végrehajtott tevékenység, de lehet egy újabb üzenetküldés is. A deaktiváció azt jelenti, hogy az adott objektum tétlen, olyan üzenetre vár, amelyet fel tud dolgozni. Az aktivációt az objektum életvonalára elhelyezett szűk téglalappal

jelölhetjük, melyet a vezérlés fókuszának nevezünk, és amelynek hossza azt az időtartamot fejezi ki, ameddig az objektum aktivált állapotban van.



Az objektumok létrejöttét többféleképpen jelölhetjük. Az egyik szokásos jelölés, hogy az objektumot jelölő téglalapot nem a diagram tetején, hanem az interakcióban eltelt időnek megfelelően lejjebb helyezzük el, és ide vezet az objektum létrejöttét kiváltó üzenet. Ezzel a jelöléssel azt is kifejezhetjük, hogy a diagram tetején elhelyezkedő objektumok, már az interakció megkezdése előtt is léteztek. Azt, hogy egy objektum az interakció során elhalálozik, az objektum életvonala mellé helyezett X-szel jelölhetjük, és ezen a ponton az életvonal is véget ér. Ha egy objektum életvonala nem fejeződik be az előbb említett módon, az azt fejezheti ki, hogy az adott objektum az interakció befejeződése után tovább él.




Felulet
UzletiLogika
1: albumLetrehozasa()
1.1: Album()
UjAlbum : Album
2: setAlbumCim()
2.1: setCim()
3: *(amig van ujabb Track)addTrackAlbum()
3.1: addTrack()
4: albumMent()
4.1: albumMentAdatbazisba()

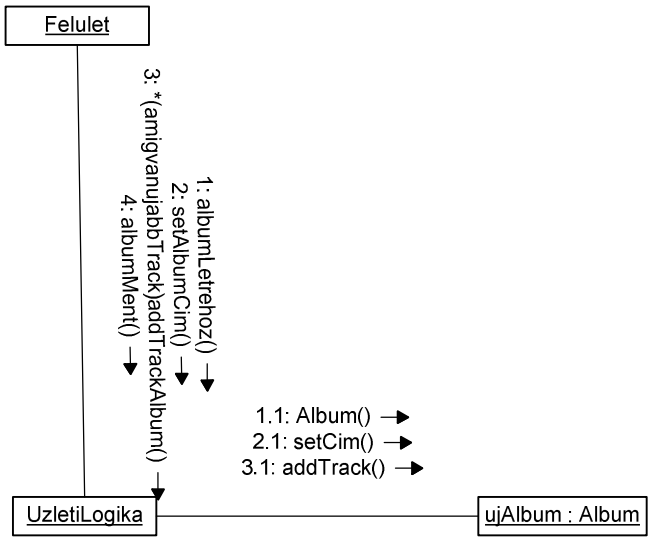
3.4 Kommunikációs diagram

A kommunikációs diagramon az objektumokat szintén az objektumdiagramtól átvett szimbólummal jelöljük, azaz téglalappal, beleírva az adott objektum nevét, elhagyva az attribútumrészt.

A diagram másik alapvető eleme a kapcsolat, amely az egymással interakciót folytató objektumok osztályai közötti asszociációk példányai, az objektumok közötti strukturális kapcsolatokat jelentik. Ezeket a kapcsolatokat az objektumokat összekötő vonalakkal modellezzük. A kapcsolatok elnevezése elhagyható abban az esetben, ha a két objektum között egyetlen kapcsolat áll fenn, egyébként az egyértelműség és érthetőség szempontjából célszerű feltüntetni őket.

Az interakciót folytató objektumok közötti üzenetváltásokat a kapcsolatokat jelentő nyilak mentén helyezhetjük el. Ez azért fontos, mert így csak azon objektumok között tudunk üzenetváltást modellezni, amelyek között létezik kapcsolat. Az kommunikációs diagram a szekvencia-diagramhoz hasonlóan támogatja a szinkron és aszinkron üzenetek használatát, valamint szinkron üzenetek esetén a visszatérés jelölését. Az üzenetek jelölése is a szekvencia-diagramnál elmondottakkal megegyező, azonban itt az üzeneteket szimbolizáló nyilakat a kapcsolatokkal párhuzamosan kell elhelyezni, jelezve ezzel, hogy az üzenetváltás az adott kapcsolaton keresztül történik. Ennél a diagramnál az üzenetek sorrendjét kötelező feltüntetni, hiszen a diagramon semmi nem fejez ki időbeliséget, sorrendet. A sorrend megadása történhet folytonos számozással, vagy hierarchikus számozással, utóbbi segítségével az egymásba ágyazott üzenetküldésekhez rendelhetünk sorrendet. Az üzenetekhez ebben az esetnem is ugyanazokat az információkat rendelhetjük, mint a szekvencia-diagramon, azaz fel kell tüntetni az üzenet nevét, opcionálisan a paramétereit és a visszatérési érték típusát, valamint lehetőség van a feltételes üzenetküldés és az iteráció modellezésére is.

Az öndelegáció esetén az üzenetet küldő és az azt fogadó objektum ugyanaz, ezért nem szükséges kapcsolatnak lenni a két objektum között. Ekkor felvehetünk az objektumhoz egy <<self>> sztereotípiájú kapcsolatot, amely az objektumból indul ki és az objektumban végződik, és így lehetőség nyílik az öndelegáció modellezésére is.



3.5 A szekvencia és kommunikációs diagramok összehasonlítása

A két diagram nagyon hasonlít egymásra, mind a modellezett fogalmak, mind azok jelölését tekintve. Tulajdonképpen mindkét diagram objektumokat, és azok interakcióit, üzenetváltásait modellezi, néhány tervezőeszköz még azt is megengedi, hogy a két diagramtípust oda-vissza konvertáljuk. Abból, hogy mindkét diagrammal az objektumok üzenetváltásait modellezzük, egyenesen következik, hogy mindkét diagram kiváló eszköz az objektumok interfész követelményeinek feltárásához. Az, hogy a küldő objektum egy üzenetet küld a fogadó objektumnak, az azt jelenti, hogy a fogadó objektumnak rendelkeznie kell azzal az interfésszel, amellyel az adott üzenetet fel tudja dolgozni, meg tudja válaszolni. Ugyanakkor fontos különbségek is vannak köztük. A kommunikációs diagram az üzeneteket az objektumok közötti kapcsolatokra vetítve jeleníti meg, azaz csak olyan objektumok között történhet üzenetváltás, amely objektumok között létezik kapcsolat. Ezzel szemben a szekvencia-diagramon tetszőlegesen, bármely objektum küldhet bármely a diagramon szereplő objektumnak üzenetet. Ez azt jelenti, hogy a kommunikációs diagram segítségével tulajdonképpen validálhatjuk az osztálydiagramokat, illetve az azon megadott asszociációkat, mivel felmerülhet egy újabb kapcsolat felvételének követelménye, illetve lehet hogy éppen a

kommunikációs diagram alapján úgy dönthetünk, hogy kihagyunk kapcsolatokat a modellből. Egy másik különbség, hogy a szekvencia-diagramon explicit módon jelölhetjük azt, hogy egy objektum az interakció során jön létre vagy éppen elhalálozik, ezzel szemben a kommunikációs diagramon egy objektum vagy szerepel, vagy nem. A szekvencia-diagram másik előnye, hogy lehetőség van az objektumok aktivációjának illetve deaktivációjának jelölésére, erre az kommunikációs diagramon nincs lehetőség, mivel ez utóbbi diagramon az idő múlását nem tudjuk szemléltetni.

3.6 Időzítési diagram

Az időzítési diagram egy speciális célú interakció diagram, amely az együttműködő objektumoknak az interakció során az üzenetek és a körülmények hatására bekövetkező állapotváltásait modellezi az időre, az objektum élettartamára vonatkozóan.

Az objektumok életvonalát ennél a diagramnál egy keretben levő téglalap alakú terület modellezi. A téglalap vízszintes tengelye az idő múlását szemlélteti, ez a tengely tetszőleges időegységekre beosztható. A téglalap bal oldalán a függőleges tengelyen mentén sorolhatjuk fel azokat az állapotokat, amelyeket az adott objektum a modellezett interakcióban felvesz. Az egymás alatt elhelyezkedő állapotok így szinteket jelölnek ki a téglalapon belül. Az állapotváltozásokat egy töröttvonal segítségével szemléltethetjük, amely azon a szinten indul, amelyhez tartozó állapotban van az adott objektum az interakció megkezdésekor, és azon a szinten ér véget, amelyhez tartozó állapotban van az objektum az interakció befejezésekor. A töröttvonal a köztes állapotok között mozog, attól függően, hogy az adott időpillanatban az objektum milyen állapotban van. Az állapotváltásokat előidéző üzeneteket a szintváltást jelző szakaszok mellett helyezhetjük el.

4 Modellmenedzsment diagramok

4.1 Csomagdiagramok

A csomagok segítségével a modellünk szorosabban összetartozó elemeit csoportosíthatjuk, magasabb szintű egységekbe szervezhetjük. A csomagokba összegyűjtött elemek lefedhetnek egy funkciókört, megvalósíthatnak egy felhasználói célt, vagy biztosíthatják a megfelelő technológiai háttért. A modell minden eleme pontosan egy csomaghoz tartozik, így a tartalmazási hierarchia egy fát alkot. A csomag egyúttal egy névterületet is jelent, amelyen belül az elemek elnevezésének egyedinek kell lenni. A csomagon belül levő elemek egymásra közvetlenül hivatkozhatnak, a csomagon kívül eső elemeket pedig minősítéssel érhetik el. Az UML a minősítőt úgy képezi, hogy az egymást tartalmazó csomagok, majd a hivatkozandó elem nevét négyponttal köti össze. A csomagokat az UML egy címkézett téglalappal jelöli, amelyen belül megadhatjuk a csomag nevét, illetve a név előtt sztereotípiát, a név után megszorítást. Az UML a csomagokhoz a következő sztereotípiákat definiálja:

- <<system>>: a hierarchia tetején álló legfelsőbb csomagot jelöli, azaz a teljes alkalmazást
- <<subsystem>>: az alkalmazáson belüli, részben független csomag
- <<modell>>: az alkalmazás egészének egy adott nézőpontból vett vázlatos képét mutatja
- <<framework>>: az alkalmazás működtetéséhez szükséges általános elemeket tartalmazza
- <<facade>>: más csomagok kiválasztott részeinek adott szempontból vett nézete, melyet általában szemléltetésként készítünk el

A csomagot jelölő téglalapban megadhatjuk a csomag lényegi tartalmát, szövegesen felsorolva vagy diagramok elhelyezésével.

Az UML definíciója szerint két modellelem között függőségi viszony áll fenn, ha az egyik definíciójának megváltozása a másik elem megváltozását is eredményezheti. Két

csomag függősége úgy értelmezhető, hogy egy a csomag által tartalmazott elem függőségi viszonyban áll egy másik csomag valamely elemével.

5 Összefoglalás

A dolgozatom célja az volt, hogy megismerjem és megpróbáljam összefoglalni az UML diagramok nyújtotta lehetőségeket, és egy egyszerű példán keresztül közérthetően bemutassam a különböző diagramtípusokat. Úgy érzem, az általam legfontosabbank vélt diagramokkal, ezek az osztálydiagram, használati eset diagram, aktivitás-diagram és szekvencia-diagram, sikerült megismerkedni, a legfontosabb fogalmakat és jelöléseket elsajátítottam. Az egyes diagramok nyújtotta lehetőségeket nem mind azonos mélységben tártam fel, azonban erre gyakran nincs is szükség, de ez gyakori a konkrét rendszerek tervezése során, hiszen az UML alkalmas csak egyes részfeladatok megoldására is, anélkül, hogy a rendszer többi elemét behatóbban ismernénk.

A dolgozatot szerintem több irányban is tovább lehetne fejleszteni: például a különböző objektum-orientált fogalmak magyarázatával, vagy a példa alaposabb kidolgozásával. Ezekkel a kiegészítésekkel kapcsolatban a dolgozatot tekintve hiányérzetem van, azonban időhiány miatt ezeket a célokat nem sikerült teljesítenem.

Végül köszönetet szeretnék mondani Pánovics Jánosnak, az egyetemi éveim és a diplomamunkám elkészítése során felém nyújtott türelméért és segítőkészségéért.

6 Irodalomjegyzék

Vég Csaba: Alkalmazásfejlesztés a Unified Modeling Language szabványos jelöléseivel, Logos2000 Kiadó, 1999

Dr. Kondorosi Károly – Dr. László Zoltán – Dr. Szirmay-Kalos László: Objektum-orientált szoftverfejlesztés, ComputerBooks Kiadó, 2004

Scott W. Ambler: www.agilemodeling.com

Angster Erzsébet: Az objektum-orientált tervezés és programozás alapjai, Kiskapu Kiadó, 1999

Tom Pender: UML Bible, Wiley & Sons, 2003