Debreceni Egyetem

Informatikai Kar

Az UML gyakorlati alkalmazásának bemutatása egy kifejlesztendő rendszeren keresztül

Témavezető:

Pánovics János

egyetemi tanársegéd

Készítette:

Szarka László

programtervező informatikus

Debrecen

2011

Tartalomjegyzék

1. Bevezetés

Egy szoftver kifejlesztése, életciklusa rengeteg apró mozzanatból, lépésből áll, s szüntelenül felmerülnek problémák, amelyeket meg kell oldanunk. Kezdve a követelményfeltárástól, egészen az implementációig, egy szoftverrendszer gondos tervezésen és dokumentáción megy keresztül, amely jelentős szerepet kap a rendszer implementálása mellett, ill. az implementáció mint életciklus előtt. A dokumentáció fontos egyrészt, mert a dokumentált követelmények, s azok állandó változásai nyomon követhetőek, a specifikáció alapján később a rendszer verifikálható, validálható, ill. természetesen magát a tervezést is megkönnyíti, a folyamatokat hatékonyabbá, átláthatóbbá teszi, hogy mikor, ki, milyen módosításokat végzett a kifejlesztendő rendszeren. Gondoljunk csak a rendszerkövetelmény-specifikációra, vagy szoftvertervezési dokumentumra, a dokumentációk hozzátartoznak a szoftverhez. Azonban nem mindegy, hogy milyen módon állunk neki egy rendszer tervezésének, mivel sok eszköz van, amelyekkel munkánkat hatékonyabbá, gyorsabbá tehetjük.

Visszatérve a problémákra, már a követelmények összegyűjtésekor felmerül az a tény, hogy a megrendelő és a fejlesztő között levő kapcsolat, nyelvezet nem elégséges, nehezen kezelhető. Dokumentációval, leíró, formalizált nyelvekkel viszont ez a kapcsolat javítható a két fél között. Egy ilyen nagyon fontos, grafikus, formalizált eszközről lesz szó ebben a szakdolgozatban, amely nem csak a szoftvertervezési folyamat külső szemmel való megértését szolgálja – amire természetesen szükség van, hiszen a megrendelőt minél jobban be kell vonni a fejlesztési folyamatban annak érdekében, hogy feltárjuk az igényeit, követelményeit, követelmények változásait – hanem azt is, hogy miként lehet a segítségével végrehajtani egy rendszer tervezését, kifejlesztését, gyakorlati alkalmazását. Ezt az eszközt fogjuk UML-nek (Unified Modeling Language) nevezni.

Ez a dokumentum az UML modellező eszköz gyakorlati alkalmazását írja le egy kifejlesztendő rendszeren keresztül. Kitér a diagramfajtákra, azok elemeire, használatára a példán keresztül, amelyek megmutatják, hogyan használhatjuk fel azt a tervezés során, viszont a rendszer kicsisége miatt nem tud átfogó képet adni minden diagram típusról, ill.

azok eszközeiről. A szakdolgozatban szereplő rendszer nem tartalmaz minden UML eszközre kiterjedő példát, viszont megpróbálja szemléltetni azt, hogy miként lehet alkalmazni ezeket az eszközöket a gyakorlatban. A szabvány UML jelölést használó diagramokat UML modelleknek nevezzük, s a különböző diagramfajták amelyeket az UML-ben használnak a rendszert különböző aspektusokból írják le, s a fejlesztés során ezek folyamatosan változnak, bővülnek, struktúrájuk változik.

2. UML (Unified Modeling Language)

2.1. Az UML-ről

Az 1990-es évekre kialakul az objektumorientált rendszerfejlesztés, azonban ezen rendszerfejlesztés mögött kezdetben nem volt olyan egyértelmű matematikai modell, amellyel a dinamizmusát értelmezni, leírni lehetett volna. Szükség volt egy olyan leíró, modellező eszközre, amely használható az objektumorientált világban.
1989-ben megalapították az Object Management Group (OMG) konzorciumot azzal a céllal, hogy az elosztott, objektumorientált rendszerek elterjedését elősegítsék. A tevékenységi köre az alapítás óta kibővült a modellalapú tervezéssel, illetve a modellalapú szabványok készítésével is. Legismertebb fejlesztései a Common Object Request Broker Architecture (CORBA), amely a heterogén környezetben működő elosztott alkalmazások fejlesztését jelentősen megkönnyíti, illetve a Unified Modeling Language (UML), amely lehetővé teszi, hogy grafikus nyelv, illetve szintaxis segítségével dokumentáljuk és modellezzük az objektumorientált rendszereket.

Az 1990-es évek elejére több mint 100 objektumorientált rendszerfejlesztési módszertan alakult ki, s ezek közül válik népszerűvé 3 módszertan, a Booch-módszer, a Rumbaugh által alkotott OMT, ill. a Jacobson-féle OOSE. Ezen három módszertanból, ill. a hozzájuk tartozó jelölésrendszerből alkottak meg egy egységes változatot, amely 1995-ben jelent meg UML 0.9 néven.

Az Unified Modeling Language (UML), egy modellező nyelv, egy elemző és tervező eszköz, amely az objektumorientált szemléletre épült. Előnye, hogy mindenkitől független, mindenki által elfogadott szabványként jött létre. Az UML egyesíti az adatmodell-koncepció (Entity Relationship Diagrams), az üzleti modellezés (work flow), az objektum-modellezés, valamint a komponens modellezés sajátosságait. Jelölésrendszere lehetővé teszi az objektumorientált fogalmak, koncepciók jelölését. Egy olyan modellező nyelvről beszélünk, amely a modellezésnél/tervezésnél alapvető szerepet játszik, és az első négy szoftverfejlesztési fázisban jut szerephez, a rendszer vázlatos leírását valósítja meg, a megvalósítással, implementálással kevésbé foglalkozik. Mivel egy nyelvről van szó, rendelkezik szintaktikai és

szemantikai szabályokkal. A szintaktikai szabályok definiálják a szimbólumrendszert, ill. azok formáit, kapcsolódási módjaikat, a szemantikai szabályok pedig az egyes szimbólumok, ill. azok kapcsolatainak értelmezését definiálják. Mindezen felül az UML egy grafikus tervező eszköz, amely diagramok segítségével ábrázolja a tervezett rendszer modelljét. Ebből adódóan támogatja a fejlesztők közötti hatékonyabb, átláthatóbb kommunikációt. Az UML a szoftveripar szereplőinek egy kész, rendelkezésre álló szabványt nyújt. Segítségével a szoftverrendszerek fejlesztői specifikálhatják, vizualizálhatják, dokumentálhatják a kifejlesztendő rendszer modelljét.
Az UML (Unified Modeling Language) szabványos, általános célú modellező nyelv, melynek segítségével szöveges és grafikus modelleket készíthetünk.

Az UML mint szabvány legfontosabb tervezési elvei:

- *modularitás*: a nyelvi konstrukciókat az UML csomagokba szervezi, nyelvi eszközeit metaosztályok példányaiként kezeli.
- *rétegzettség*: a rétegek elkülönítik a példányokat különböző absztrakciós szinteken.
- *particionálás*: a csomagokat partíciókra ossza, a szakterülethez igazítást segíti.
- *kiterjeszthetőség*: UML-profilok hozhatók létre, ez két értelemben történhet: kiterjeszthető *platformspecifikus* profil felé (pl. van .NET-profil, EJB-profil, J2EE stb.) profilok hozhatóak létre szakterületekre (pl. pénzügy, telekommunikáció, stb.) és harmadik, ezektől független értelemben az UML mint nyelv kiterjeszthető, ezáltal *UML-nyelvcsalád* létezik, az alapszabvány elemeinek felhasználásával.
- *újrafelhasználhatóság*: alapvető tervezési szempont volt, hogy elemei tetszőlegesen újrafelhasználhatóak.

2.2. Történeti áttekintés

- 1980-as évek végére több objektumközpontú modellezési irányzat is kialakult. Némelyiket elsősorban (objektumorientált) programok modellezésére használták, másokat egyéb célokra, például adatbázis-tervezésre.
- 1990-es évek elején három modell különült el. Ezek kifejlesztői Jim Rumbaugh, Ivar

Jacobson és Grady Booch voltak.

- 1990-es évek közepén a Rational Software alkalmazta Rumbaugh-t és Booch-t, akik megalkották az Unified Method 0.8-at. 1996-ban már Jacobson együtt dolgozták ki az Unified Method 0.9-et, valamint megalakul az UML Partners konzorcium, az UML támogatására, amelyhez a legnagyobb informatikai cégek, úgymint a Microsoft, Oracle, HP, DEC stb. csatlakoztak.
- 1997-ben már Unified Modeling Language néven adták be az OMG (Object Management Group) nevű független szabványszervezethez, mely azóta is fejleszti.
- 1998: UML 1.2
- 1999: UML 1.3
- 2000: UML 1.4
- 2003: UML 1.5
- 2004: a jelenlegi UML 2.0- verzió elkészül.

3. A víziótól az osztálydiagramig

3.1. Első lépések

Kis projektek esetén sokkal kevesebb osztályból épül fel a rendszer, így az elemzési osztály diagramban is kevesebb osztály fog szerepelni, nagyobb összetettebb rendszereknél sok osztály fog előfordulni ennélfogva majd a megvalósítás, tervezési fázisba való átültetés során nagyobb valószínűséggel fognak megjelenni problémák.
A tervezés elején a szakterületi fogalmak összegyűjtésével indulunk el, s a kezdetben megalkotott tervezési diagramtól haladunk a megvalósítási osztálydiagram felé, miközben a diagramunkat folyamatosan alakítjuk.

Kezdetben a fogalmainkat, osztályainkat fogjuk jelölni az elemzési osztály diagramban.
Az elemzési osztály diagramban fogalmi osztályok jelennek meg, s szakterületi fogalmakat próbálnak leírni. Maga az elemzési osztály diagram szakterületi fogalmak elemeit tartalmazza. A fogalmak mellett megjelenhetnek bizonyos értelembe vett technikai aspektusok, valamint megjelenhet a megvalósítási szint is, amelyben adott programozási nyelvekhez kapcsolódó információkat, dolgokat lehet majd hasznosítani.

Első lépés egy szoftver kifejlesztésénél, a követelmények meghatározása, elemzése, de tételezzük fel, hogy a követelményeket már meghatároztuk, s az osztálydiagram létrehozása következik. Az elemzés során felmerül a kérdés, hogy a követelmények elemzése alapján, hogy lehet megalkotni az adott problémához tartozó osztálydiagramot. Általában kétféle követelményleírással találkozunk. Ha a leírás matematikailag jól meghatározott, akkor a diagram elkészítése viszonylag könnyű feladat. Ha a leírás nem formális, akkor a mondatok analizálásával juthatunk eredményre. Erre használhatunk egy ökölszabályt, amely az esetek többségében hatékonyan alkalmazható. Először megpróbáljuk elkülöníteni a szövegben a probléma szempontjából lényeges főneveket, amelyek az osztályoknak felelnek meg, majd a kapcsolatokat derítjük fel a leírás alapján.

A szoftverfejlesztési folyamatok során a felhasznált eszközök, ill. tervezésre és specifikálásra fordított idő nagy mértékben befolyásolja a fejlesztés hatékonyságát. Kisebb projekteknél a tervezés sokkal kevesebb időt vesz igénybe, előfordul, hogy ez a lépés ki is marad a szoftver életciklusából. Azonban nagyobb, komplexebb rendszereknél elengedhetetlen a modellezés és a megfelelő specifikáció megtervezése, hogy a fejlesztési folyamat sikeres legyen.

3.2. A kifejlesztendő rendszer

A kifejlesztendő rendszer egy mp3 lejátszó, amelynek életciklusának első négy lépését követhetjük végig az elemzéstől a tervezésig haladva, mely során az UML mint modellező nyelv lesz a segítségünkre. Ennek gyakorlati alkalmazását fogjuk bemutatni a rendszerünk kifejlesztése során. A kifejlesztendő rendszerről egyfajta kezdetleges víziót olvashatunk az alábbiakban, mely megfelel egy rövid kezdetleges felhasználói követelménynek.

A kifejlesztendő rendszer egy mp3 lejátszó, amely mp3 fájlok lejátszására alkalmas.
Egy névvel ellátott lejátszási lista van a rendszerben, amelyhez hozzáadhatunk egy vagy több, vagy egy egész könyvtárnyi mp3 fájlt, vagy esetleg egy egész, már korábban elmentett lejátszási listát.
A támogatott formátumok, mp3 esetében: *.mp3, lejátszási listák esetében: *.m3u,*.pls. (ezek felismerése automatikusan történik).
Miután az mp3 fájlok bekerülnek a rendszerbe, azok a listában jelennek meg, rájuk kattintva lehetőség van meghallgatni őket.
Az mp3 fájlok ID3v1tag-je is szerkeszthető.
Módosíthatjuk a számok listában levő sorrendjét, új számok hozzáadására, ill. korábbiak közüli törlésre is van mód, ezen felül lehet a listában levő elemeket rendezni, bizonyos paraméterek szerint.
Egy adatbázis, minden újonnan megnyitott mp3-at regisztrál magának, hogy később könnyen elő lehessen belőle keresni.
Az adatbázisból név szerint kikereshetünk számokat, a találatok közül hozzáadhatunk a listánkhoz.
Az adatbázishoz lehetőség van még egy egész könyvtárnyi mp3-at hozzáadni.

3.3. Az UML diagramjai

A szabvány UML jelölést használó diagramokat UML modelleknek nevezzük. Az UML diagramok eltérőek, a rendszert gyakran különböző aspektusból szemlélik, azonban az egyes diagramok között lehetnek átfedések, megtörténhet, hogy adott tervezési lépés előfordul több diagramon is, de más nézőpontból, a fejlesztés során ezek folyamatosan változnak, bővülnek, struktúrájuk változik.

- Statikus szempont szerint:
    - Osztálydiagram (Class): a rendszer objektumelvű szerkezetének leírása.
    - Objektumdiagram (Object): az osztálydiagram egy példányát mutatja be.
- Dinamikus szempont szerint:
    - Állapotdiagram (Statechart): azt mutatja meg, hogy a rendszer milyen állapotokon keresztül, milyen állapotátmenetekkel oldja meg a feladatot.
    - Szekvenciadiagram (Sequence): az objektumok közötti üzenet váltások időbeli menetét szemlélteti.
    - Aktivációs diagram (Activity): a tevékenységek és az objektumok egymásra gyakorolt hatását fejezi ki (vezérlések, rendszerfunkciók).
    - Együttműködési diagram (Collaboration): az objektumoknak a probléma megoldásában való együttműködését mutatja be.
- Implementációs szempont szerint:
    - Komponensdiagram (Component): a komponensekből felépített szoftverrendszert mutatja be.
    - Alrendszerdiagram: az alrendszerek kapcsolatát írja le.
- Környezeti szempont szerint:
    - Konfigurációs diagram (Deployment): a szoftverrendszer környezetének, a hardver-szoftver konfigurációinak a szemléltetésére szolgál.
- Felhasználói szempont szerint:
    - Felhasználói esetek diagramja (Use case): a rendszernek és felhasználóinak kapcsolatát adja meg.

A fejlesztés során ezek közül azt használjuk, amely a tervezési folyamat szempontjából szükséges, említésre méltó. Azokat, melyekkel modellezni tudjuk a szoftver részeit a tervezés során, azonban ez a fejlesztett rendszertől függ. Lesznek olyan diagramok, amiket nem használunk fel a tervezéskor, mert más diagramokkal ugyanazokat a részeit a rendszernek már modelleztük, dokumentáltuk. A dolgozatban sem térünk ki mindegyikre, csak azokra, amelyeket a legfontosabbnak tartunk, azonban ezekre majd ki fogunk térni a fejlesztés egyes szakaszaiban.

3.4. Megszorítások

Korábban a bevezetőben volt szó arról, hogy az UML mint nyelv kiterjeszthető, ezáltal *UML-nyelvcsalád* létezik, az alapszabvány elemeinek felhasználásával. A kiterjeszthetőség lehetővé teszi, hogy az UML jelölésrendszerét specializáljuk a fejlesztés által meghatározott irányba. Modelljeinket kiegészíthetjük úgy, hogy a szakterület vagy az alkalmazott technológia megfelelő jelölései a szabvány keretein belül maradnak.

Háromféle kiterjesztési mechanizmusa van az UML-nek:

- *sztereotípia*: minden konstrukcióelemhez adható sztereotípus, <<>> között, nevének megadásával, amely vagy pontosítja az adott elemet vagy a kiterjesztésben játszik szerepet, vagy új modellelemet hoz létre. A szabványban rengeteg beépített sztereotípia szerepel.
- *megszorítás*: minden elem mellett megadható, olyan általános eszköz, mellyel a modellünket pontosítani tudjuk, annak olyan jellemzőit adhatjuk meg, amelyeket másképpen nem lehet kifejezni. Olyan körülményekről van szó, amely a modellelemre vonatkozik, s annak teljes életciklusa során fenn áll. Jelölése: {}
- *kulcsszavas értékek*: ezekkel a modellelemek specifikációját egészíthetjük ki explicit módon, név-érték párok segítségével. Jelölés: a modellelem mellett név=érték formában, kapcsos zárójelek között.

3.5. Osztálydiagram

3.5.1. Az osztály

Az objektumorientált szemlélet központi fogalma az osztály. Tervezés során a valós világon hajtunk létre egy absztrakciót, vagyis a programozás egy adott szintjén a megoldás szempontjából lényegtelen részeket elhanyagoljuk, a fontosakat kiemeljük. A középpontban a programozási nyelvek absztrakciós szintjének növelése áll, ami által a valós világ könnyebben modellezhetővé, a problémák könnyebben megoldhatóbbakká válnak. Az egységbezárás elve alapján a valós világot egyetlen modellel kell leírni, és ebben kell kezelni a statikus és dinamikus jellemzőket, tehát, az adatmodell és a funkcionális modell egymástól elválaszthatatlan. Ahogy összegyűjtjük a valós világ lényeges jellemzőit, azokat osztályokba próbáljuk sorolni. Az osztály az absztrakt adattípust valósítja meg, rendelkezik attribútumokkal és metódusokkal.

Az UML rendszerfejlesztési fázisaiban való alkalmazása során az osztálynak 4-féle közelítése lehet:

- *fogalom* (követelményfeltárásnál): egy szakterületi fogalom absztrakciója jelenik meg egy osztályban.
- *típus* (elemzésnél): absztrakt adattípus annak összes tulajdonságával együtt.
- *objektumhalmaz* (adatbázis-kezelésnél, OO-adatmodellezésnél): tipikusan ebben a közelítésben szuperosztály/alosztály.
- *implementáció*: a hatékony implementációt segíti elő a modellezés.

Az **osztály** fogalma az osztálydiagramok alapja. A modellezés során határozzuk meg az osztályok nevét, attribútumait és műveleteit. Az osztályok jelölése a diagramon egy téglalappal történik, s 3 részre bontható:

- Az első részben szerepel **a név**, amelynek egyedinek kell lennie, nagybetűvel kezdődik, középre igazított, s körülötte sztereotípiák/megjegyzések, stb. szerepelnek. Ezt kötelező megadni, s egyedinek kell lennie, mert ezzel fogjuk azonosítani a rendszer egy elemét.

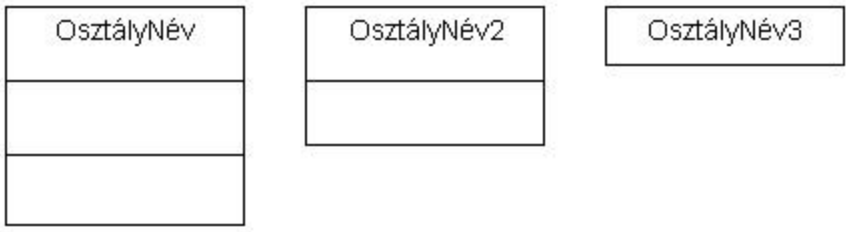
- A második részben szerepelnek **az attribútumok**, melyek formája a következő:
  név [: típus] [= kifejezés]
  Sem a típus, sem a kifejezés nem kötelező, a név kisbetűvel kezdődik. Ha származtatott az attribútum, akkor azt a neve előtti /-jel mutatja, ha pedig osztályszintű, akkor azt aláhúzással jelöljük.
- A harmadik részben **a műveletek** (metódusok) szignatúrája van megadva a következőképp:
  név (paraméterek) [: típus].
  A név ugyanúgy jelöljük mint az attribútum nevet, a paraméterek között vessző van. A paraméter alakja:
  [mód] név: típus[= kezdőérték].
  A mód lehet IN/OUT/INOUT/RETURN. Az IN az alapértelmezett, a RETURN visszatérési paramétert jelent.
  Az UML nem foglalkozik a metódusok implementációjával, interfészekkel foglalkozik, melyekkel hozzáfér a metódusokhoz, s ezeket műveleteknek nevezi.

3.1. ábra

Minden attribútum és művelet előtt vagy megjelenhetnek bezárás/láthatósági szintek, vagy láthatósági szekcióba rendezem az adott attribútumokat, műveleteket a szekció alapszó utáni felsorolással. Az egyes szintek értelmezése megegyezik az objektumorientált paradigma láthatósági szintjeinek értelmezésével. Ezek a következők, jelöléssel együtt:

- [+] *Public*: az attribútumot látja az összes osztály.
- [#] *Protected*: az attribútumhoz csak a leszármazott osztályok férhetnek hozzá.
- [-] *Private*: az attribútum csak az adott osztályban használható.
- [~] *Package*: az attribútum az osztályt tartalmazó csomagból hivatkozható.

A diagramokban az osztály ábrázolásánál, jelölésénél a nevét, az attribútumok neveit, esetleg a műveletek absztrakt formáját tüntetjük fel. Gyakran azonban a modellalkotás során egyszerűbb formákat használunk a terv áttekinthető ábrázolásának érdekében:



3.2. ábra

3.5.2. Relációk

- Asszociáció:

  A legáltalánosabb reláció két osztály között az asszociáció. Ez két osztály közötti absztrakt reláció, amely kétirányú társítást fejez ki. A reláció azért absztrakt, mert a reláció konkretizálása az osztályok objektumainak összekapcsolásával valósul meg. Konkrét esetben ez összekapcsolás, absztrakt esetben, társításról beszélünk. Osztályok között egy folytonos vonallal jelöljük. Fontos megjegyezni, hogy a relációban az osztályok objektumai közül általában több példány is részt vehet.
  A vonalnak két vége van, ezek megnevezhetők szerepkörökkel, ez a kapcsolattípussal való totális azonosság, van számossága, amelyet a vonal adott vége felett jelöljük 1-gyel, *-al, vagy intervallummal n..m alakban.
  Az asszociációhoz tartozik két beépített megszorítás:

- {ordered}: meghatározott sorrendben érhetőek el az adott osztály objektumai.
- {sorted}: a példányok a kapcsolatban rendezettek.

Az asszociációnál jelölhető a navigálhatóság a folytonos vonal végén nyitott fejű nyíllal, amely azt jelenti, hogy az egyik osztályból, hogy lehet eljutni a másikba. Ennek letiltását a vonal végén X-el jelöljük.

- Aggregáció:

  Az osztályok közötti speciális „egész-rész viszony”. Ezt a folytonos vonal végén egy üres rombusszal jelöljük. Olyan aszimmetrikus viszonyt jelöl, amelyben az egész oldali műveletek megjelennek a rész oldalon, vagyis az aggregáció azt fejezi ki, hogy az egyik osztály objektumai részét képezik egy másik osztály objektumainak.

- Kompozíció:

  Ez egy speciális aggregáció, az egész-rész viszony speciális esete. Tömött rombusszal jelöli az UML, s azt jelenti, hogy a rész oldal nem élheti túl az egészet. Az aggregációs objektum és annak komponensei azonos életciklusban léteznek, azaz egyszerre jönnek létre és egyszerre szűnnek meg.

3.6. Az elemzés

Az objektumelvű modellezés során a követelmények leírásából a megoldás három modelljét állítjuk elő. A statikus modell az osztálydiagramokból és a hozzájuk tartozó osztályleírásokból, valamint az objektumdiagramokból és az ezekhez tartozó objektumleírásokból áll. Azaz: statikus modell = osztálydiagram+osztályleírások, objektumdiagram+objektum leírások.

Az osztálydiagram megalkotásának lépései:

1) A követelmények elemzése alapján meghatározzuk azokat a főneveket, amelyek potenciálisan objektumjelöltek lehetnek.
2) Objektumok leírása, tulajdonságjellemzők meghatározása. Ez tartalmazza a tulajdonságok, műveletek és relációk felderítését.
3) Objektumok osztályba sorolása hasonló tulajdonságaik alapján.
4) Osztályok közötti relációk meghatározása.
5) Kezdeti osztálydiagram megszerkesztése.
6) Az objektumok attribútumainak, a relációk tulajdonságainak, szerepeknek, multiplicitásoknak meghatározása.
7) Általánosítás segítségével az osztálydiagram hierarchikus szerkezetének kialakítása, áttekinthetőségének növelése, egyszerűsítése.
8) Osztályleírások elkészítése.

A kifejlesztendő rendszer követelmény-leírása:
A kifejlesztendő rendszer egy **mp3 lejátszó**, amely mp3 fájlok lejátszására alkalmas.
Egy névvel ellátott **lejátszási lista** van a rendszerben, amelyhez hozzáadhatunk egy vagy több, vagy egy egész könyvtárnyi mp3 fájlt, vagy esetleg egy egész, már korábban elmentett lejátszási listát**.**
A támogatott formátumok, mp3 esetében: *.mp3, lejátszási listák esetében: *.m3u,*.pls. (ezek felismerése automatikusan történik).

Miután az **mp3 fájlok** bekerülnek a rendszerbe, azok a listában jelennek meg, rájuk kattintva lehetőség van meghallgatni őket.
Az mp3 fájlok **ID3v1tag**-je is szerkeszthető.
Módosíthatjuk a számok listában levő sorrendjét, új számok hozzáadására, ill. korábbiak közüli törlésre is van mód, ezen felül lehet a listában levő elemeket rendezni, bizonyos paraméterek szerint.
Egy **adatbázis**, minden újonnan megnyitott mp3-at regisztrál magának, hogy később könnyen elő lehessen belőle keresni.
Az adatbázisból név szerint kikereshetünk számokat, a találatok közül hozzáadhatunk a listánkhoz.
Az adatbázishoz lehetőség van még egy egész könyvtárnyi mp3-at hozzáadni.

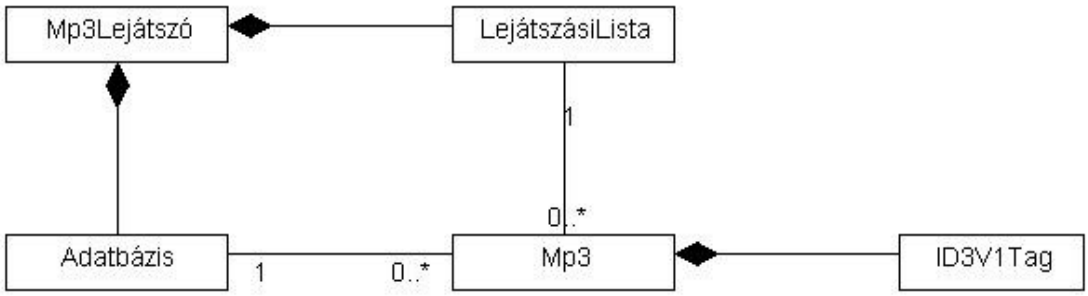
Látható, hogy milyen komponensekből fog felépülni a rendszer. A következő osztályokat gyűjtöttük ki: mp3 lejátszó, lejátszási lista, mp3 fájl, id3v1tag, adatbázis.

Ezen osztályok között derítjük fel a kapcsolatokat. Hogyan kapcsolódnak ezek egymással?

- **relációk**: A lejátszó és a lejátszási lista között egy kompozíciós kapcsolat van, amely egy tartalmazást jelöl, miszerint az mp3 lejátszó tartalmaz egy listát, annak a része. A lejátszási lista vagy üres, vagy egy, vagy több mp3 fájlt tartalmaz. Egy vonallal kötjük össze az mp3-at és a listát, ezek között egy asszociációs relációt definiálunk. Minden mp3-nak van 1 db Id3v1tag-je, vagy ha nincs akkor majd a rendszer generál egyet automatikusan, ez szintén egy kompozíció, hiszen a tag része az mp3-nak, ha az mp3 megszűnik, az Id3v1tag is. Végül az adatbázis is több mp3-at tartalmaz, az adatbázis és az mp3 között egy asszociáció van.
- **szerep**: az Mp3 lejátszó a vizuális megjelenítésért felelős, ez bizonyos szempontból a felhasználói felület szerepét tölti be.
- **multiplicitások**: Mivel 1 lejátszónk van, s egy listánk, így a multiplicitást nem szükséges jelölni a kettő közt, ugyanis a vonal, amely összeköti az osztályokat alapértelmezésben mindkét végén 1-1 kapcsolatot jelent, ha nem írunk semmit. De ettől függetlenül mindkét oldalra írhatunk egy 1-est. A lejátszási lista és az mp3 fájl közötti viszonyban a lista több mp3-at is tartalmazhat, ezt úgy jelöljük, hogy a reláció

mp3 felőli oldalán feltüntetjük a 0..* jelöléssel a számosságot. Természetesen a kapcsolat másik végén 1-es szerepel, mivel 1 db listánk van. Az Id3v1tag és az Mp3 között szintén nem fontos jelölni, de kiírhatjuk, hogy azok 1-1 kapcsolatban vannak. Végül az adatbázis is több mp3-at tartalmaz, az mp3 felőli oldalon 0..* számosság jelenik meg.

- **attribútumok**: hogy melyik osztály milyen tulajdonságokkal rendelkezik, azt majd később fogjuk tárgyalni.



3.3. ábra

3.7. Adattagok és viselkedésmódok

Láttuk, hogy az osztálynak 3 része van, az UML-ben vannak a nevén kívül attribútumai és műveletei, de ezeket nem tüntettük fel. Ez sokszor előfordul, mert könnyebb az áttekintés, s több osztályt tudok felrajzolni ugyanakkora méretben, hogyha kihagyjuk ezeket az információkat. Természetesen a bővítés során ezeket is meg fogjuk adni.

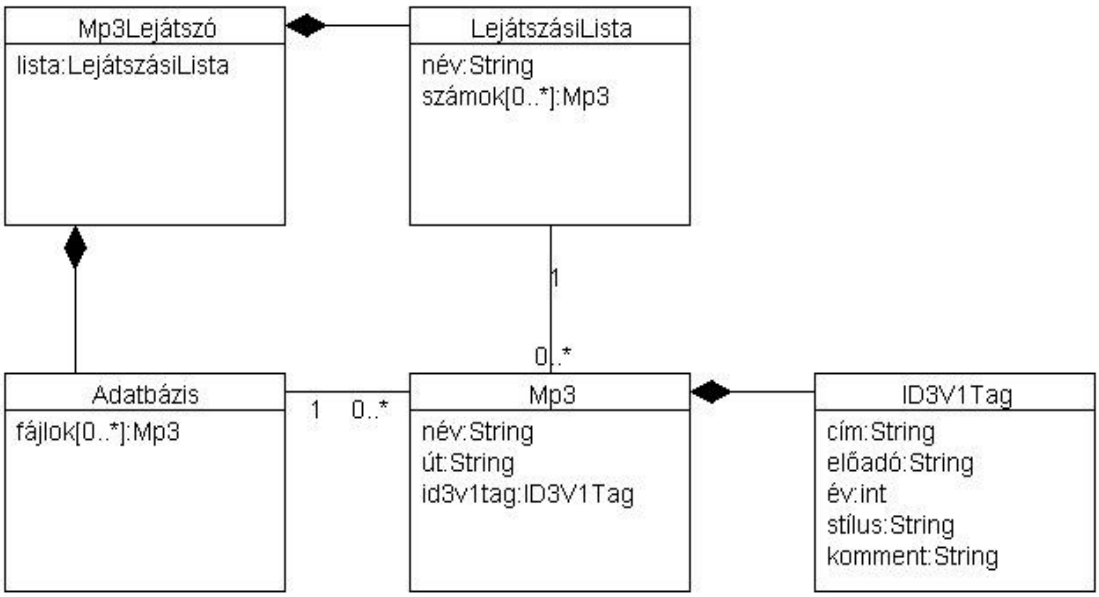
A modellezés egy kreatív folyamat. Ez egyszerre előny is és hátrány is. Az, hogy bizonyos dolgokat miként modellezek, az sok esetben a modellező szemléletmódjára, megközelítésére van bízva. Ez azért előnyös, mert a modellező viszonylag így nagy szabadsággal rendelkezik, azonban ez hátrány is, mert bizonyos értelemben könnyű hibázni. Az, hogy miben lehet a modellezésben hibázni, vagy hogy mi a rossz modell, ill. hogy mikor mondjuk, hogy egyik modell jobb mint a másik, az releváns. Nehéz mérhető minőségű számokat, objektív irányelveket meghatározni.

Ezen kérdések az attribútumok meghatározásánál vetjük most fel, ugyanis az osztálydiagram kialakításához közeledve minden osztálynak megnézzük milyen tulajdonságai, attribútumai vannak. A fentiekhez kapcsolódóan felvetjük, hogy pl.: az Mp3 a valós világ egy létező egyede a Lejátszási listában, amely tulajdonképpen a rendszerünkben egy valódi fájlt fog azonosítani, reprezentálni, egy külön adattípusként jelenik meg. És mondhatjuk azt, hogy az Id3v1tag mint az mp3 felvétel adatait tartalmazó osztály nem annyira lényeges mint az mp3. Mert ez Előadó, Cím, Stílus stb. attribútumokkal rendelkezik és az esetek túlnyomó többségében azt gondolnánk, hogy ezen attribútumok adatmodellezési szemszögből nézve nem egy-egy új egyedtípusként jelenik meg mint az Mp3. Ez legtöbbször így van, de a modellező gondolhat arra is, hogy az előadó is egy létező személy, egy entitást, amely egy új típusként jelenhet meg, s az előadóról is tárolhat a rendszer információkat, függetlenül az Id3v1-tag-től. Az, hogy melyik a jó megközelítés, objektív irányelvek híján nehéz megfogalmazni.

Minden osztálynak megnézzük milyen tulajdonságai, attribútumai vannak. Tisztáztuk, hogy a lejátszó tartalmazza a lejátszási listát és az adatbázist. Ez a kezdetleges osztálydiagramról leolvasható, hogy mi mit tartalmaz. Az mp3 osztály attribútumok mezőjében mint típus jelennek meg a lejátszási lista és az adatbázis osztályok. A lejátszási listának van neve, ill. maga a lista, ami a számokat tartalmazza. Az mp3-ainknak szintén van név attribútuma, egy út attribútum, amely a fájl helyét jelöli a merevlemezen, ill. van egy Id3tag-je. Az Id3v1-tag-ben található attribútumok közül vegyük a legegyszerűbbeket: Előadó neve, Számcím, Év, Stílus, Komment. Mindezeket az osztályok attribútum mező részben feltüntetjük. Ezek, mint a diagramok többi része később újabb diagramváltozatoknál változhatnak az elemzés, tervezéstől függően.

Az osztályok attribútumainak és a metódusainak meghatározása a modellezőtől függ. Az elemzési osztálydiagramban, mikor fogalmi szinten beszélek a dolgokról, gyakorlatilag típust sosem mondok, mert nem érdekes, s a reprezentáció sem az, hogy milyen jellegű adatról van szó, hiszen az a lényeg, hogy pl.: az Mp3-nak van egy neve, egy elérési útja, ill. egy Id3v1tag-je, s hogy pl.: a neve egy 64 bit hossz számsor, vagy legfeljebb. 20 hosszúságú link, az

igazából egy absztrakt fogalmi jelentése, amely itt még nem fontos.



3.4. ábra

A metódusok meghatározásához szintén visszatekintünk a kezdetleges követelményleíráshoz. Tisztáztuk azt, hogy ha a leírás nem formális, akkor a mondatok analizálásával juthatunk eredményre, s ekkor nehezebb dolgunk van. Itt szintén a korábban, hasonlóan alkalmazott ökölszabályt használhatjuk, amelyet az osztályok, objektumok meghatározására használtunk. Itt viszont a mondatokban az állítmányokat elemezzük, hogy mi mit hajt végre, s ezek alapján derítjük fel, hogy melyik feladat, művelet, melyik osztály felelősségéhez tartozik. A legtöbb ilyen feladatot mint fogalmat szintén érdemes felvenni a fogalomszótárba. A követelményleírás alapján:

Az Mp3 lejátszó mp3 fájlok lejátszására alkalmas. A fogalomszótárba definiálnunk kell, hogy hogy működik az mp3 lejátszás. Így jutunk el oda, hogy az mp3 lejátszó feladata a felvételek lejátszása, a felvétel szüneteltetése, megállítása, következő és előző felvételre ugrás. Ezek a feladatok tartoznak a lejátszóhoz, s ezek megjelennek metódusokként az osztályban.

Kiegészítés: ezen kívül ha lejjebb merülünk az implementációs réteg felé, akkor azt is kijelenthetjük már itt is, hogy a grafikai megjelenítés is legyen a lejátszó feladata, de erre még itt nem feltétlen lenne szükség.

Vizsgáljuk a többi osztályt is. A lejátszási listának van egy neve, s egy listája teli mp3-akkal. Neki az mp3-ak kezelése a feladat. A lejátszási lista felelősségéhez tartozik új fájlok hozzáadása, törlése, mozgatása, lista betöltése, mentése, rendezése, s mindez a követelményleírásból olvashatóak. Ezeket részletesebben elemezzük, kiderül pl.: hogy a lista mentése is 3-féleképp történhet, m3u-ba, pls-be vagy txt-be, ez mind-mind külön metódusként jelenhet meg a diagramban.

Kiegészítés: itt a fájlok listában való mozgatása, hogy miképpen történik, az implementációs kérdés, ezzel itt nem fontos foglalkoznunk.

Kiegészítés: nagyon fontos megfigyelnünk, hogy az egyes feladatok külön osztályokhoz tartoznak. Lehet olyan modellezést is végrehajtani, hogy pl.: nem hozunk létre Lejátszási Lista osztályt, hiszen elég a nevét és egy listát az Mp3 lejátszóban mint attribútumokat tárolni, s ez esetben minden metódus amit itt most a Lejátszási lista metódusaiként definiálunk, azok mind a Lejátszó hatásköréhez fognak tartozni. Miért nem vonjuk össze a listát vagy akár még az adatbázist is a lejátszóba?
Érdemes minél jobban szétbontani a modellt már a tervezés elején, hogy később megkönnyítsük a dolgunkat a refactorálás során. Az ilyen viszonylag előre gondolkodás a modellező sajátossága, gyakorlati úton sajátíthatóak el. Ha sok osztályt a példánkban összevonnánk, akkor később az implementáció során könnyen olyan antimintákba kerülhetünk, melynek a kijavítása nagyon problémás, esetleg nagyobb rendszereknél lehetetlen újratervezés nélkül.

Az adatbázis fizikai tárolása, és feladatai is későbbi, implementációs szinten megvitatható kérdéseket vet fel. Az elemzés során csak annyit tudunk, hogy mp3-akat fog tárolni, könyvtárakat lehet hozzáadni, mp3-akat lehet keresni, s magát az adatbázist kitisztítani,

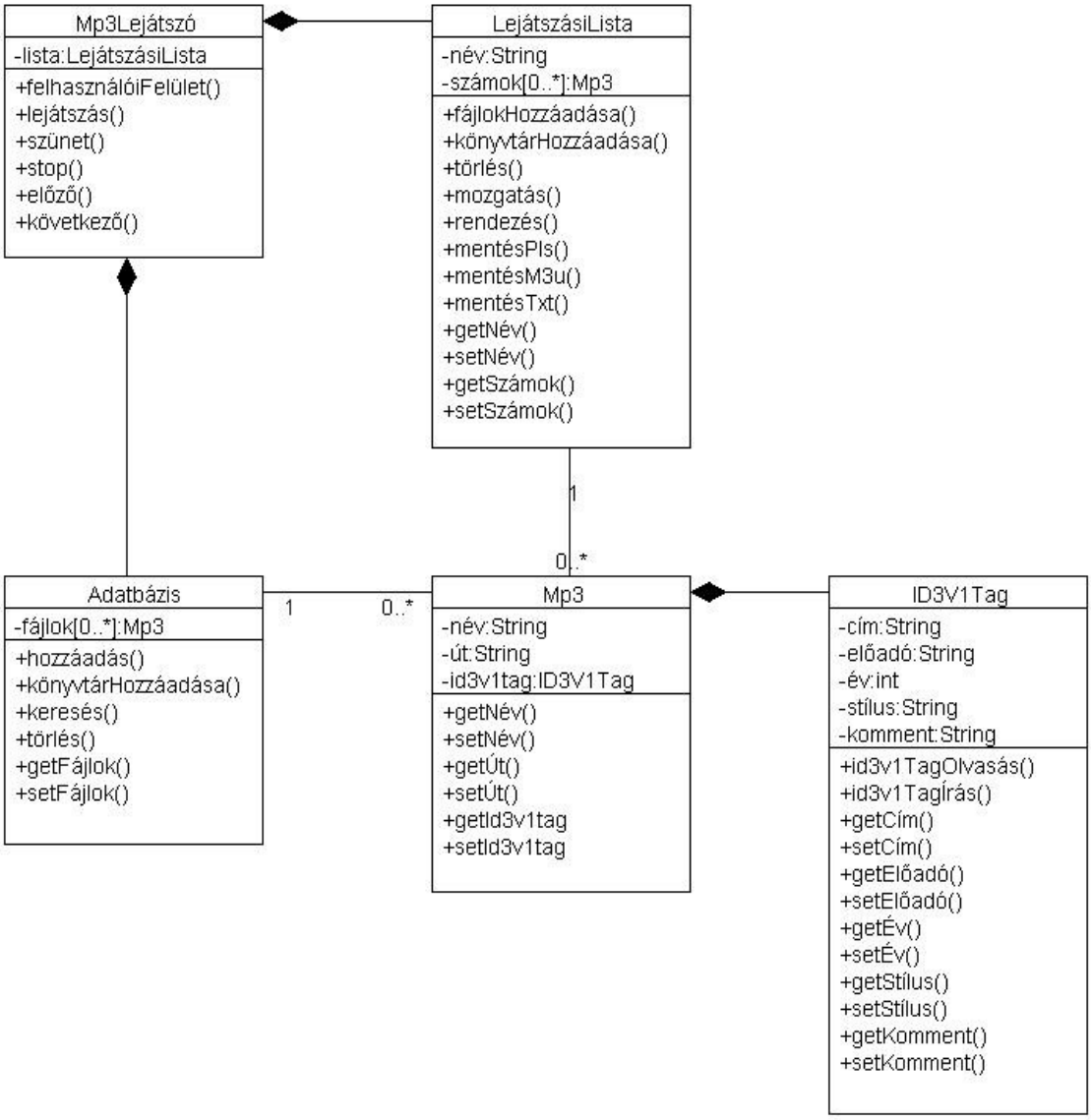
törölni lehet.

Az mp3-nak különös feladata nincs, az Id3-tag-nek is annyi, hogy kell egy olyan metódus, ami majd a fizikai mp3-ból kiolvassa az Id3tag-et, ill. írja azt, de ezzel is majd később, a megvalósítási szinten kell foglalkozunk.

A metódusok meghatározása ugyanolyan kérdéses probléma mint az attribútumoké. Összegyűjtjük a feladatokat, amelyeket az egyes osztályok hajtanak végre, de a tényleges megvalósításukkal az UML nem foglalkozik, ezért nem kell jelölnünk. Miután az attribútumok és a metódusok is megvannak, már csak egy lépés választ el minket a kész osztálydiagramtól.

Leírtuk a fogalmi szintű viszonyokat. A következő lépés, az elemzési osztálydiagramból a tervezési osztálydiagramba való átlépés. A tervezési osztálydiagram az tipikusan az, ahol már az eddig felvett attribútumok mellé meghatározzuk azok láthatóságát. Ezek olyan dolgok, amik általános objektumorientált fogalmak. Egy adott adattagról nem teljesen nyelvfüggő, hogy mi a láthatósága, de nem is teljesen fogalmi szintű. A láthatóságok meghatározásához a tervezés során le kell ereszkednünk az általános fogalmi szint és az implementációs szint közé. Itt megjelenhet valamilyen típusra vonatkozó információ is.
Mivel a példa adattagjainak privát láthatóságot állítunk be, ezért olyan metódusok is megjelenhetnek, amelyek ezek elérését, beállítását végzik.

Kiegészítés: jelen példában minden adattag privát láthatósággal rendelkezik, de ha már a megvalósítási szint felé közeledünk, akkor észre vehetjük, hogy pl.: az adatbázis és a lejátszási lista elemeit a lejátszó gyakran fogja használni, s ez esetben könnyíthetünk azok elérésén, pl.: ha a lejátszási lista, lista attribútumának vagy az adatbázis fájlok attribútumának csomag szintű láthatóságot adunk, azonban ez szintén modellező függő kérdés.


Mp3Lejátszó
-lista:LejátszásiLista
+felhasználóiFelület()
+lejátszás()
+szünet()
+stop()
+előző()
+következő()
LejátszásiLista
-név:String
-számok[0..*]:Mp3
+fájlokHozzáadása()
+könyvtárHozzáadása()
+törlés()
+mozgatás()
+rendezés()
+mentésPls()
+mentésM3u()
+mentésTxt()
+getNév()
+setNév()
+getSzámok()
+setSzámok()
1
0..*
Adatbázis
-fájlok[0..*]:Mp3
+hozzáadás()
+könyvtárHozzáadása()
+keresés()
+törlés()
+getFájlok()
+setFájlok()
1
0..*
Mp3
-név:String
-út:String
-id3v1tag:ID3V1Tag
+getNév()
+setNév()
+getÚt()
+setÚt()
+getId3v1tag
+setId3v1tag
ID3V1Tag
-cím:String
-előadó:String
-év:int
-stílus:String
-komment:String
+id3v1TagOlvasás()
+id3v1TagÍrás()
+getCím()
+setCím()
+getElőadó()
+setElőadó()
+getÉv()
+setÉv()
+getStílus()
+setStílus()
+getKomment()
+setKomment()

3.5. ábra

4. Felhasználói esetek és forgatókönyvek

4.1. Az üzleti folyamat

Röviden szót ejtünk a rendszerünk üzleti folyamatáról, annak leírásának egy változatáról. Ezt mind csak érintőlegesen, s nézünk egy példát az üzleti folyamat táblázatos leírására.
Ehhez meg kell határoznunk, hogy mi az a folyamat:

- A folyamat, egy strukturált, mérhető tevékenységsor, amelynek célja egy speciális termék (vagy szolgáltatás) előállítása adott fogyasztó vagy piac számára.
- A folyamat a munkatevékenységek speciális sorrendje, kezdő és végponttal, világosan meghatározott bemenetekkel és kimenetekkel, idő és térbeli tényezőket figyelembe véve, azaz: a működés struktúrája.
- A folyamatnak van költség- és idővonzata, az eredményének van minősége, és kapcsolódik hozzá fogyasztói megelégedettség.

A rendszerfejlesztés során az elemzés fázisban tervezzük meg a folyamatot, hogy a rendszer milyen lényeges pontjain megyünk keresztül, amelyek azt a célokat, lényeges komponenseket tartalmazzák, amiért a rendszert kifejlesszük. A példánk egy mp3 lejátszó, s azt a célt szolgálja, hogy mp3 fájlokat nyissunk meg vele, azokat meghallgassuk, a beépített gyors-keresővel mp3-akat keressünk a gépünkről egy adatbázisból, ill. a végén a lejátszási listát elmentsük. Ezek a rendszer lényeges részei, mérföldkövei a rendszer üzleti folyamatát tekintve.
Az üzleti folyamatot táblázatos módszerrel fogjuk leírni, amelynek említésre méltó mezőinek jelentése:

- *Név*: Az üzleti folyamat neve.
- *Rövid leírás*: A folyamat tartalmának rövid leírása.
- *Érintett aktorok*: Az aktorok határrendszerek, amik az adott folyamathoz kapcsolódnak. Különbséget tehetünk elsődleges, az üzleti folyamatra hatással levő aktorok, és másodlagos, pusztán az üzleti folyamat lebonyolításában részt vevő, de rá hatással nem lévő aktorok között.

- *Kiváltó esemény*: Olyan esemény, amely hatására elindul a folyamat. A dialógusfolyamatok esetén ezek tipikusan a felhasználók cselekményei, vagy olyan események, amelyek a felhasználókhoz kapcsolható módon következnek be, vagy nekik tulajdoníthatók.
- *Előfeltétel*: A rendszer állapotára szabott feltétel, amelynek teljesülnie kell ahhoz, hogy az üzleti folyamat sikeresen lefuthasson. Ha ez a feltétel nem teljesül, és a folyamat ennek ellenére elindításra kerül, az utófeltétel teljesüléséről, ill. az esetlegesen fellépő hibákról semmilyen megállapítást nem tehetünk.
- *Standard lefutás*: A standard lefutást az elsődleges forgatókönyv névvel is illetik. Ez a normális esetet írja le, vagyis az elviekben leggyakoribb eset, esetleg legegyszerűbb, eredetileg előforduló eset.
- *Kivételek és alternatívák*: Az elsődleges forgatókönyvtől eltérő esetek, ezek a másodlagos forgatókönyvek. Minden eltérő esetre vonatkozóan megállapításra kerül az előfordulás gyakorisága és további lefutás.
- *Utófeltételek*: Ha az előfeltétel teljesülése mellett bekövetkezett az elsődleges vagy valamely másodlagos s forgatókönyv, ez a rendszer állapotára vonatkozó feltétel garantáltan teljesül.
- *Eredmény*: A folyamatnak a résztvevők számára értékelhető eredmények.
- *Gyakoriság*: A gyakoriság meghatározása kiindulási alapként szolgál az üzleti folyamatok feldolgozáskori prioritásának meghatározására. Gyakran hasznos még várható gyakoriságot feltüntetni a kivételek és alternatívák rovatba.
- *Utalások*: Általában folyamatok leírására hivatkozik más dokumentumokra is.
- *Megjegyzések, nyitott kérdések*: A minta ezen mezője minden egyébre használható, ami a többi mezőbe nem illik bele. Hasznát vehetjük például nyitott kérdések, részletek regisztrálására.

Példa táblázatos módszerrel leírt üzleti folyamatra:

Név	
Mp3 lejátszó	
Rövid leírás	
Mp3 fájlokból egy listát hozunk létre	
Érintett szereplők	
Egy, a rendszert használó felhasználó	
Kiváltó esemény	
A felhasználó egy válogatást szeretne készíteni mp3 fájlaiból	
Előfeltétel	
A felhasználónak legyenek mp3 felvételei	
Standard lefutás	Kivételek és alternatívák
1. A felhasználó megnyit mp3-akat, vagy egy lejátszási listát 2. Belehallgat az mp3 felvételekbe 3. Módosítja az adott listát, hozzáad, töröl belőle, megváltoztatja a számok sorrendjét 4. További fájlokat keres a beépített adatbázisból 5. Elmenti a listát, amit majd később használni akar.	a) Fájl megnyitási probléma, nem találja a keresett fájlt. b) Nem sikerül elmenteni a listát c) Nincs még felvétel az adatbázisban
Utófeltétel	
Legyenek a lejátszóban felvételek betöltve	
Eredmény	
Létrejön a kívánt lista	
Gyakoriság	
Alkalmanként, használattól függ, hetente, naponta akár többször	
Utalások	
Mp3 lejátszók, Id3v1tag, Lejátszási listák	
Megjegyzés	
Nincs	

4.6. ábra

Röviden meghatározhatjuk, hogy ezekből mi a konkrét üzleti folyamatunk:

- fájlok megnyitása
- lejátszás
- lista kezelése
- keresés
- kimentés

Illetve a lényegesebb kérdések, amelyeket ezek szerint majd tisztáznunk kell:

- Fájlok megnyitása: Milyen fájlokat? hogyan?
- Lejátszás: Hogyan és miket játszunk le?
- Hozzáadás: Miket adunk hozzá?
- Szerkesztés: Milyen információkat adunk meg?
- Törlés: Miket törlünk?
- Keresés: Mit keresünk, honnan, hogyan frissítjük az adatbázist?
- Kimentés: Hogyan, hova, milyen formátumba mentünk?

4.2. Use Case

Az osztálydiagram, az osztályok és azok attribútumainak, ill. metódusainak meghatározását követően most a felhasználói szempont szerinti felhasználói esetdiagram létrehozását fogjuk elvégezni, amely külső személy számára könnyen érthető, átfogó képet ad a kifejlesztendő rendszer funkcionalitásairól. Ez a diagram típus egy viselkedési diagramok csoportjába tartozik, amelyek azt írják le, hogy minek kell történnie a modellezett rendszerben. Észrevesszük, hogy a funkciók, bizonyos értelemben a követelményleírásból kielemzett osztályok metódusaiként szolgálnak valamilyen szempontból, így hogy mivel azokat már lejegyeztük, a következő diagram elkészítése könnyebb feladat lesz.

A felhasználói esetek diagramja a felhasználók szempontjából kívánja szemléltetni azt, hogy a rendszer miként működik, függetlenül attól, hogy a szolgáltatásait hogyan valósítja meg. Tehát azt szemlélteti, hogy mit tud a rendszer, függetlenül attól, hogy azt miként hajtja végre.

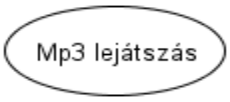
A rendszer által nyújtott szolgáltatásokat együtt mutatja be azokkal, akik ezekkel a szolgáltatásokkal kapcsolatba kerülnek.

Ez a diagram eszköze annak, hogy a felhasználók és a rendszer fejlesztői a rendszerrel szemben támasztott követelményeket azonos módon értelmezzék:

A diagram részei:

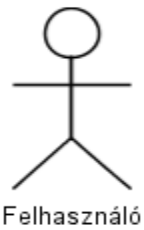
- felhasználói esetek,
- felhasználók,
- felhasználási relációk.

A **felhasználói esetek** a rendszer funkcióinak összefoglalásai, szolgáltatási egységek. Ez az egység az akcióknak egy olyan sorozata, amelyekkel a rendszer a felhasználók egy csoportjával működik együtt. Pl.: egy információs rendszerben ilyen lehet a számlamozgásokról történő szolgáltatás az ügyfelek számára.



4.7. ábra

A felhasználók az adott rendszeren kívüli egységek, más programrendszerek, alrendszerek, osztályok, ill. személyek lehetnek. Ezek **aktor** szerepet töltenek be.



4.8. ábra

A felhasználói relációkat a felhasználás módjának alapszava egészítheti ki, alapszavak felhasználásával.

A felhasználói esetek diagramja az áttekinthetőség céljából rendszerint szintekre tagolt.

Relációk:

- *Asszociációk*: Folytonos vonallal jelöljük, a használati eset és az aktor között van. Itt is jelölhető a számosság.
- *Általánosítás* (generalization): Üresfejű nyíllal jelölhetjük, használati eset és használati eset, vagy aktor és aktor között szerepel.
- *Include*: <<include>> vagy <<uses>> sztereotípiával ellátott reláció. Két használati eset között áll fent, ha az egyik magába foglalja a másikat. (az egyik használati eset használja, és mindig használja a másikat).
- *Kiterjesztés*: <<extend>> jelölésű reláció, kibővítés (kivételkezelés, hibakezelés). Az egyik használati eset működését egy másik eset kiegészíti. Pl.: a „jelszóellenőrzés" használati esetet kibővíthetjük egy olyan funkcióval, amely lekezeli azt, ha a felhasználó hibás jelszót ad meg.) Bizonyos folyamatoknál vannak fontos résztevékenységek, ilyenkor szokás ezeket is leválasztani és az eredeti kiterjesztéseként felfogni.

A rendszerrel szemben támasztott követelmények összegyűjtésének, és a használati eset diagram kialakításának legfontosabb lépései:

- A rendszerhez kapcsolódó az aktorok összegyűjtése. Gyakran használt ökölszabály a főnevek keresése a szöveges specifikációból: Kik a rendszer felhasználói? Ki felel a rendszer karbantartásáért? Mik a rendszer által használt erőforrások? Mik a rendszerhez kapcsolódó más rendszerek?
- A rendszer viselkedését, funkcióit kifejező használati esetek összegyűjtése. Ökölszabály: igék keresése a specifikációból: Mire használják a rendszer? Mit csináljon a rendszer? Hogyan használják a rendszert? Mit tudjon a rendszer?
- Az aktorok és a használati esetek kapcsolatainak vizsgálata, pontosítása. Ökölszabály: igék és főnevek közti kapcsolatok alapján.

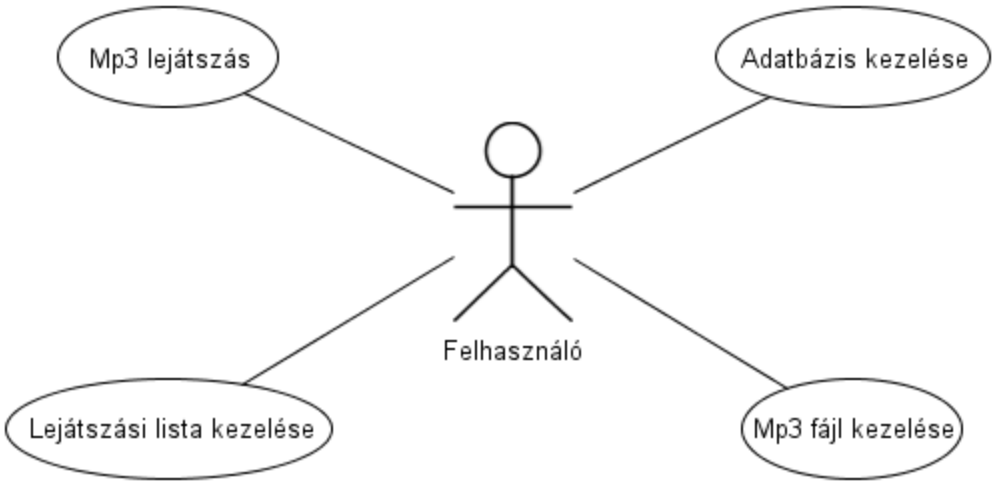
Ahhoz, hogy megállapítsuk a rendszer legalapvetőbb funkcióit, össze kell állítanunk a kifejlesztendő rendszer üzleti logikáját. Ezt már a fejlesztés legelején megcsináltuk, de a példánkban leginkább a felhasználói esetek diagramjához kapcsolódik, ugyanis, a felhasználói

esetek diagramjához általában tartoznak bizonyos dokumentumok, amelyek konkrétan a használati esetek megvalósítását jelentő viselkedéseket magyarázzák (narratívák), valamint olyan szöveges dokumentumok, amelyek az egyes használati esetek forgatókönyvét írják le, amely egy használati eset egy lehetséges megvalósulását jelenti.

Visszatérünk az elemzési fázishoz, amikor a követelményeket feltártuk, s kiderítettük, hogy mik azok az elemek, az ún. aktorok, amelyek a rendszeren kívül esnek, s amelyek egy bizonyos szerepkört fejeznek ki. A mi rendszerünkben egy aktor szerepel, egy felhasználó fogja használni a rendszert. A felhasználó olyan módon kapcsolódik a rendszerhez, hogy használja annak funkcióit. A kérdés, hogy milyen funkciókat használ, ill. konkrétabban milyen funkciókon keresztül kapcsolódik a felhasználó a rendszerhez?

A követelmények elemzése során a rendszer viselkedését, funkcióit elemezzük. Ezek lesznek a bizonyos használati esetek, amelyek leírják, hogy miként, hogyan használja a felhasználó a rendszert, pontosabban milyen funkciókat használ. A jelenlegi példánknál az üzleti folyamat, a követelmény leírás, az osztálydiagram metódusai mind-mind segítenek a funkcionalitások kiderítésében, miszerint a rendszerünkben a felhasználó felvételeket játszhat le, lejátszási listát kezelhet, kereshet felvételeket (ami együtt jár a beépített adatbázis kezelésével), ill. bizonyos értelemben szerkesztheti (az Id3v1tag módosításával) az mp3-akat is.

A rendszer jellege miatt könnyű helyzetünk van, de egy nagyobb rendszernél, ahol több aktor vesz részt a használatban, komolyabb probléma az aktorok és a használati esetek (funkciók) közötti kapcsolatok kialakítása. A négy legfőbb funkcionalitást összegyűjtve megalkotjuk a használati eset diagram vázát, majd pontosítsuk a diagramot a többi kisebb funkció bevonásával.



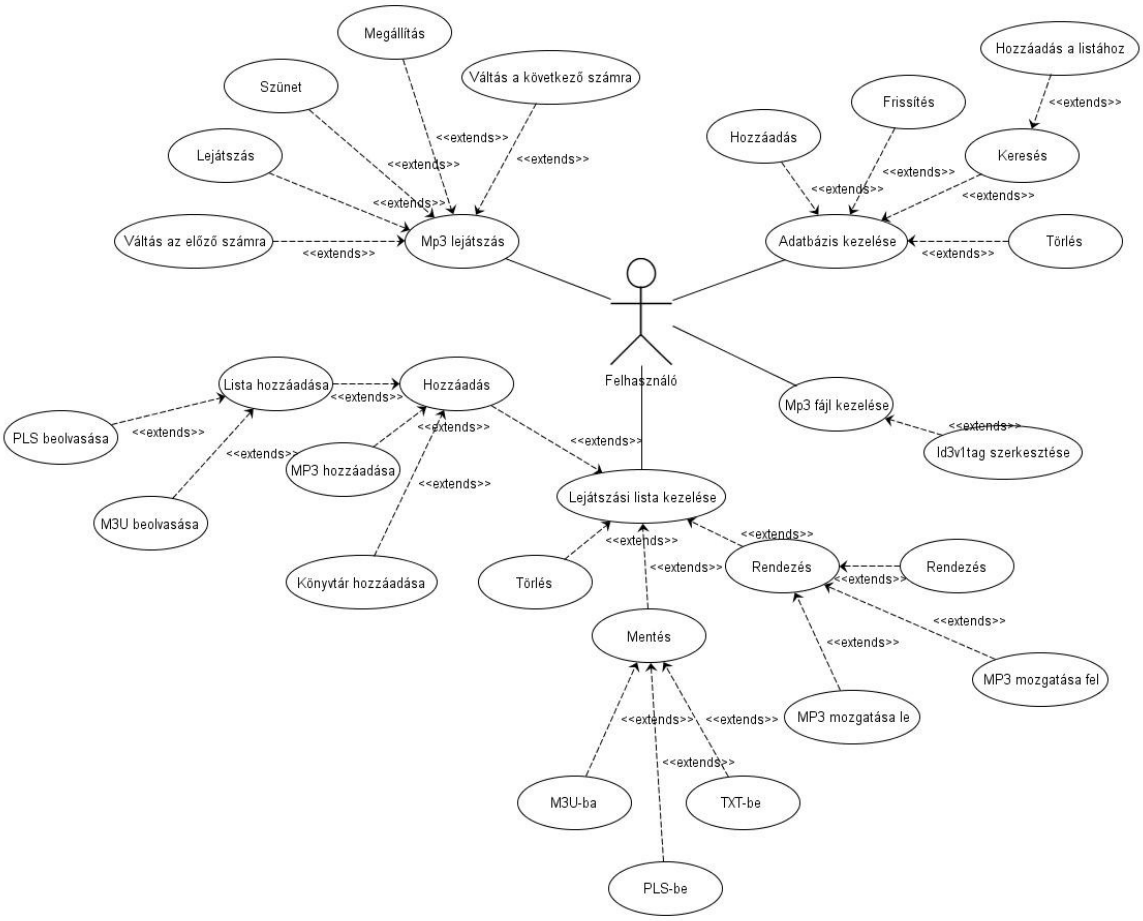
4.9. ábra

A specifikációból kigyűjthetünk kérdéseket, amelyek közelebb visznek minket az esetek összegyűjtéséhez. Az üzleti folyamat és a folyamatokkal kapcsolatos kérdések:

- fájlok megnyitása
- lejátszás
- lista szerkesztése
- keresés
- mp3 kezelés
- kimentés

- Fájlok megnyitása: milyen fájlokat? hogyan?
- Lejátszás: hogyan és miket játszunk le?
- Lista szerkesztése (menedzselés):
    - Hozzáadás: miket adunk hozzá?
    - Szerkesztés: milyen információkat adunk meg?
    - Rendezés: mi alapján rendezzük?
    - Törlés: miket törlünk?
- Keresés: mit keresünk, honnan, hogyan frissítjük az adatbázist?
- Kimentés: hogyan, hova, milyen formátumba mentünk?

Többé-kevésbé a használati esetek kidolgozása is a modellező szemléletmódjától függ, hogy mit veszünk össze, s mit bontunk szét. Jelen alkalmazásban az Mp3 lejátszás az egyik fő funkció, amelyet kibonthatunk olyan módon, hogy a lejátszás, szünet, megállítás, és a számok közötti váltás mind egyazon eset kiterjesztése, részfunkciói.
A következő esetünk a lejátszási lista kezelése, amelyet tovább bonthatunk a listához való hozzáadásra, listából való törlésre, lista elmentésére, ill. a lista elemeinek rendezésére. Tulajdonképpen a fájlok megnyitása, ill. a kimentés, ahogy arról korábban említést tettünk a lejátszási lista felelősségéhez tartozik. Ennél a diagramnál hasonló okok miatt vesszük ezeket az eseteket ide.
Az mp3-ak kezelése egy külön használati eset, amely mint Id3v1Tag kezelésként jelenik meg. Végül a keresés funkció tulajdonképpen egy bővebb fogalom hatáskörébe tartozik, mert ugye korábban már kiderítettük, hogy egy adatbázisból fogunk keresni. Lehetőségünk van különvenni az adatbázis kezelésének esetét, ill. a keresés használati esetet, de most a bővebb, adatbázis kezelésének esetén belülre rakjuk a keresést, szintén a korábbiak miatt. Lényegében szemlélettől függ, hogy ezt miként alkotjuk meg.


Megállítás
Szünet
Váltás a következő számra
Lejátszás
Váltás az előző számra
Mp3 lejátszás
<<extends>>
Hozzáadás a listához
Frissítés
Hozzáadás
Keresés
Adatbázis kezelése
Törlés
Felhasználó
Lista hozzáadása
Hozzáadás
PLS beolvasása
MP3 hozzáadása
M3U beolvasása
Könyvtár hozzáadása
Lejátszási lista kezelése
Törlés
Mentés
Rendezés
Rendezés
MP3 mozgatása le
MP3 mozgatása fel
M3U-ba
TXT-be
PLS-be
Mp3 fájl kezelése
Id3v1tag szerkesztése

4.10. ábra

5. A dinamikus modell és az aktivitási diagram

A modellezendő rendszerünk fejlesztése során beletekintettünk a statikus modell alkotóelemeibe, valamint a felhasználó szempont szerinti diagramtípusba. A következő lépés az objektumelvű modellezés egy másik fontos elemének a rendszer dinamikus modelljének leírása. A dinamikus modell három fő részből tevődik össze: az állapotdiagramokból, az állapotok leírásából és az események leírásából. Az események leírására szolgál a szekvencia, együttműködési és aktivációs diagram. A szekvencia és az együttműködési diagram az objektumok interakcióját határozzák meg. Az osztály objektumainak dinamikus viselkedését a probléma megoldása során az állapotdiagram írja le, ami egy állapotautomatát ábrázol. A jelenlegi példánkhoz a fentebb említett diagramok közül az aktivitási és szekvencia diagramokat fogjuk felhasználni a modellezéshez.

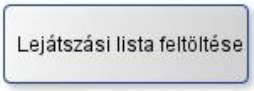
5.1. Aktivitási diagram

Az aktivitási diagram a hagyományos adatfolyam-diagram egy módosított változata. Az aktivációs diagram a probléma megoldásának lépéseit szemlélteti, a párhuzamosan zajló vezérlési folyamatokkal együtt. Az állapotdiagram egy változatának is tekinthető, amelyben az állapotok helyére a végrehajtandó tevékenységeket tesszük, és az átmenetek a tevékenységek befejezésének eredményeként valósulnak meg. Két fajta aktivációs diagramot szokás megkülönböztetni: az életfolyam alapút és a sávos alapút.

Az aktivitási diagram legfontosabb feladata az alkalmazás dinamikájának, valamint az időben lezajló változások szemléltetése. Képes nem csak egymás utáni, hanem egymás melletti tevékenységek, párhuzamos tevékenységek ábrázolására. Akkor használjuk, amikor valamilyen folyamatot akarunk modellezni. Ez lehet részrendszerek együttműködése, használati eset működési módja, részrendszeren belüli használati esetek együttműködése, vagy algoritmus leírása.

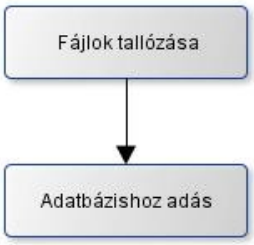
Alapelemei:

- *Aktivitás*: Egy végrehajtandó tevékenységet jelent, lehet egy lényeges algoritmus, egy osztályon belüli tagfüggvény, egy vagy több utasítás, melyeknek a végrehajtása időbe telik. Ezeket ívelt oldalú téglalappal jelöli az UML. A tevékenység általában nem atomi, további al-tevékenységekre bontható, ezeket adott esetben újabb aktivitási diagramon modellezhetjük.



5.11. ábra

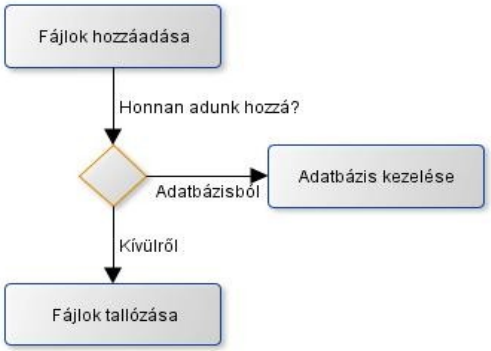
- *Átmenet*: A nyílhegyben végződő vonallal jelölt, végrehajtandó tevékenységek időbeli sorrendje, ahol a nyíl hegye az adott tevékenység befejezése utáni következő tevékenység felé mutat.



5.12. ábra

- *Szinkronizációs vonal*: Amikor nem lényeges a sorrend, akkor jelölhetjük egymás melletti vagy konkurens folyamatok jelölésére. Olyan ÉS kapcsolat, amely szálakra bontja a vezérlést, amelyek nem feltétlenül párhuzamos végrehajtásúak, csak logikailag függetlenek. Ha nem szükséges a párhuzamosítás, akkor tetszőleges sorrendben egymás utáni végrehajtás történik. Vastag vízszintes vonallal történik a jelölése. A következő végrehajtás, minden ág sikeres végrehajtás után következhet.

- *Döntés*: Alternatív tevékenységcsoportok közötti választás, elágazás, amelynek több kimenő éle is lehet. Tulajdonképpen ez egy VAGY kapcsolat. A döntés kivezető éleit őrszemekkel címezhetjük.



5.13. ábra

- *Kezdő- és végállapot*: Az alkalmazás kezdetét és végét jelölik. A használatuk nem kötelező, pszeudo állapotoknak is nevezik. A végállapotból több is lehet, míg kezdő állapotból csak egy. A kezdőállapot jelölése egy körlappal, a végállapot egy körben levő körlappal történik.

5.14. ábra

5.2. A rendszer céljai

Az aktivitási diagram tulajdonképpen az üzleti modellezésben is használt folyamatábra UML-beli megfelelője. Nem meglepő tehát, hogy a kialakításához először visszatekintünk az üzleti folyamatunkra, hogy mi a program célja:

- mp3-akat adunk a listánkhoz
- módosítjuk a listában levő mp3-ak sorrendjét
- elmentjük a listát

Megvizsgáljuk mi hogyan aktiválódik, mit miért teszünk? Összegyűjtjük az ok-okozati tényezőket:

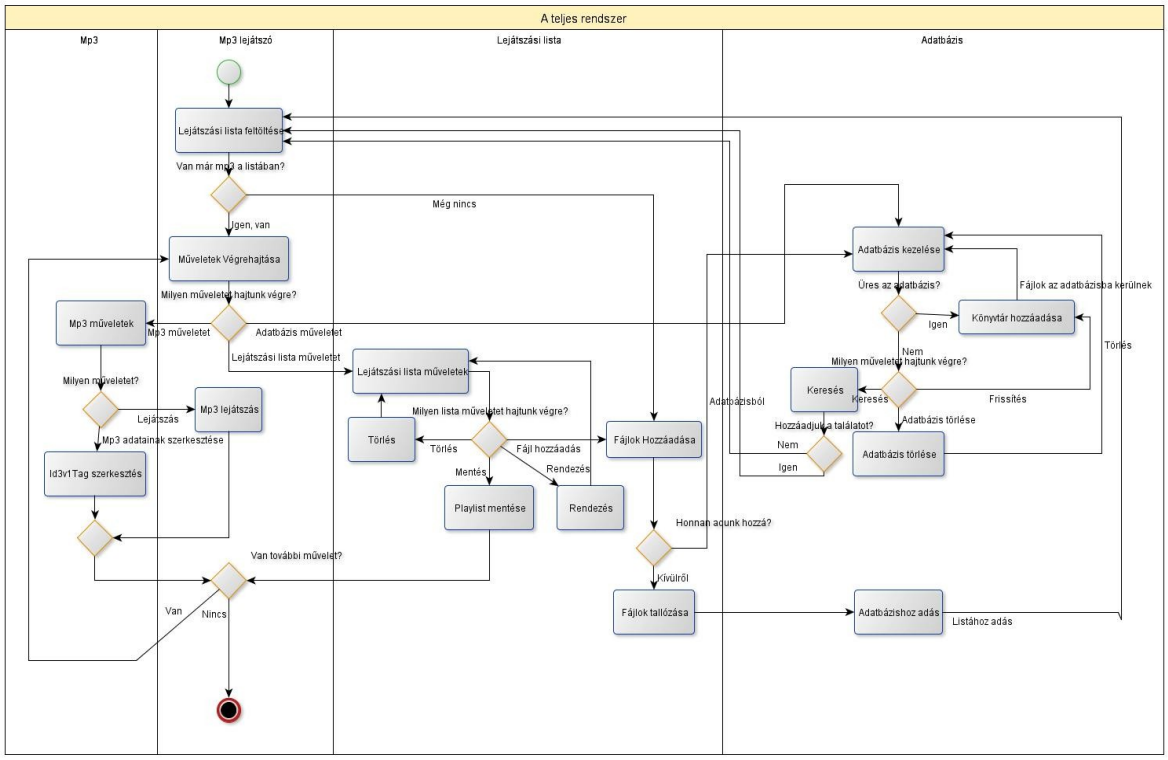
- Mp3-akat adunk a listához, azért, mert a lista legelőször üres, persze aztán később is lehet, de ez elengedhetetlen, hogy legyen egy lista amit készítsünk.

Kibővítés: fájlok kívülről, ill. az adatbázisból adhatóak hozzá a listához, keresés útján.

- A program célja, hogy egy listát készítsünk, ehhez a felvételekbe bele kell hallgatnunk, s ez alapján módosítani azok helyét, ill. rendezni a listát.
- A listánk elmentésével az üzleti folyamatunk befejeződik.

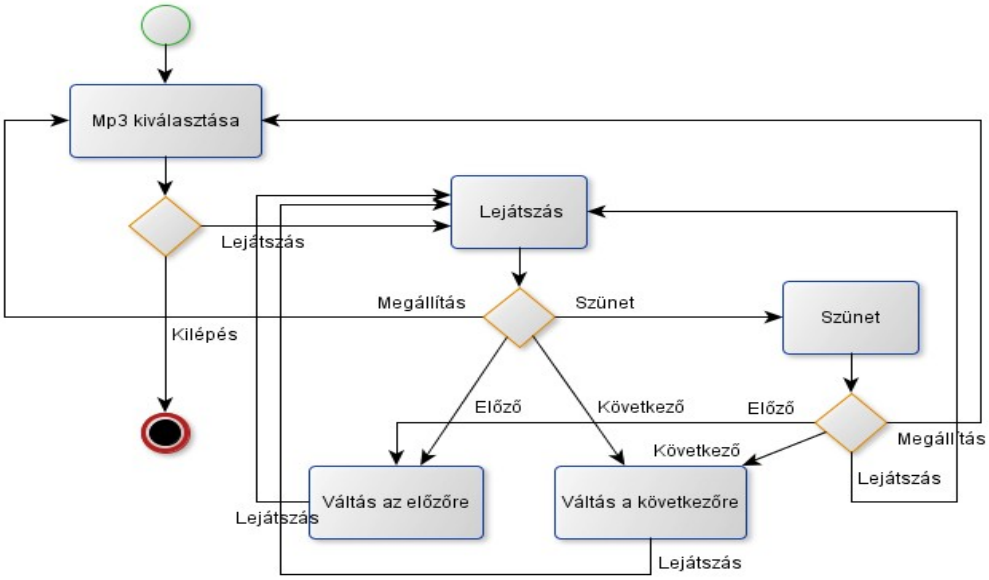
Ezen három funkciót vesszük alapul az aktivitási diagram elkészítéséhez.
Van először is egy olyan funkció, ahol az mp3-akat kezeljük, egy olyan, amivel a lejátszási listát, s egy olyan, amivel az adatbázist. Különválogatjuk a különféle eseményeket. Először is kezdjük a lista felépítésével.

Kérdés, hogy vannak-e már felvételek a listában? Amennyiben nincsenek, ez esetben szükség van fájlok hozzáadására. Ezt kétféleképp tehetjük meg, vagy az adatbázisból rákeresünk számokra, s azok közül adunk hozzá, vagy a merevlemezről töltünk be mp3-akat, list fájlokat, könyvtárakat.
Az adatbázishoz kapcsolódó rész következik, amikor az adatbázisból keresünk számokat. Itt az a kérdés, hogy van-e már betöltött fájl az adatbázisba? Ha nincs, akkor hozzá kell adnunk valamely könyvtárat, amely mp3-akat tartalmaz. Persze ezt akkor is megtehetjük, ha már vannak az adatbázisban elemek. Ezek után már lehetőségünk van keresni az adatbázisból, majd a találatok közül számokat hozzáadni a listához. Ezenkívül persze az adatbázis teljesen ki is üríthetjük egy törlés esemény segítségével.
Ha kívülről adunk hozzá elemeket, akkor minden addig nem ismert felvétel nem csak hogy a lejátszási listába kerül, de az a adatbázisba is, hogy később ha rákeresünk, egyből az adatbázisból megtaláljuk.
Ha már vannak mp3-aink a lejátszási listában, jöhetnek a különféle műveletek az mp3-akon és a listán.
A felvételeinkkel a következőket tehetjük, lejátszhatjuk őket az mp3 lejátszóval, ill. az Id3v1tag-üket módosíthatjuk.
A lejátszási listával kapcsolatos műveletek a listához hozzáadás, listából való felvétel törlése, lista rendezése, és a legutolsó, a lista elmentése.
Ezek után a tranzakciókat, műveleteket megismételhetjük, a végén pedig, ha már kész vagyunk, s a feladatainkat elvégeztük, kilépünk a rendszerből.



5.15. ábra

A felső ábrán látható, hogy a diagramot felosztottunk aszerint, hogy melyik tevékenység minek a felelősségi köréhez tartozik. Ez azért van, mert az aktivitási diagram alapesetben nem tartalmaz olyan információt, hogy melyik tevékenység mihez tartozik, s ezzel a módszerrel lehet ezt jelölni. Ezek után a sávokhoz objektumokat, aktorokat rendelhetünk. Ez segítséget nyújt a tervezés további menetéhez. Egyrészt könnyebb az osztályokhoz tartozó attribútumokat, ill. azok egymással való kapcsolataikat beazonosítani, azonban ténylegesen ezzel majd csak a következő diagram fog foglalkozni. Itt még arra térünk ki, hogy az egyes eseményeket, új diagramon lehetőségünk van részletezni, kibontani. Jelen példánkban a felvétel lejátszását mutatjuk meg egy külön ábrán, annak egyfajta konkrétabb, algoritmikusabb leírásán keresztül. Ilyen módszerrel pontosíthatjuk, részletezhetjük, ill. a fentebb mutatott felosztással kategorizálhatjuk az eseményeket a tervezés során.


Mp3 kiválasztása
Lejátszás
Lejátszás
Megállítás
Szünet
Szünet
Kilépés
Előző
Következő
Előző
Megállítás
Következő
Lejátszás
Váltás az előzőre
Váltás a következőre
Lejátszás
Lejátszás

5.16.ábra

6. Kommunikációk és szekvenciák

A viselkedési diagramok segítségével nagyjából leírtuk, magas szinten megfogalmaztuk a követelményeinket. Ezeket a követelményeket használati esetek segítségével tudtunk leírni. Egy használati eset tipikusan egy tevékenység sorozatot jelentett. Az, hogy ez a folyamat milyen lépésekből fog állni, hogy ebben milyen objektumok vesznek részt, azt még nem feltétlen tudjuk, így ezekből még nem készíthető el a rendszer kódja, implementálása. Ennek érdekében kellene leírnunk, hogy az egyes diagramunkban pl.: használati eset diagramban, de az oda be nem kerülő folyamatokat, tevékenységet igénylő dolgok külön zajlanak, s ezeket a viselkedési diagramokkal tudjuk leírni. Ezen viselkedési diagramok egymásnak sok esetben versenytársaik, előfordulhat, hogy ugyanazt a tevékenységet két különböző diagramtípussal nincs értelme leírni, mert bizonyos értelemben hasonló eredményre fogunk jutni. Azonban az egyikből olyan dolgokat tudhatunk meg, ami a másikból nem fog kiderülni, az egyik bizonyos szempontból jobban szemlélteti az adott modellezendő részét a programnak mint a másik. Az eszközrendszerből is látszani fog, hogy milyen módon lehet az adott diagramtípust megválasztani.

A szekvenciadiagramok azt mutatják be, hogy a tevékenységben részt vevő objektumok hogyan fognak egymással kommunikálni, interakcióba lépni. A kommunikációjuk egy interakció sorozatnak fogjuk tekinteni, és ez által felosztjuk a felelősségi köröket, hogy melyik objektum milyen felelősséggel bír, s ezek segítségével majd új metódusok jöhetnek létre, s látni fogjuk, hogy maga metódus lánc, hogy fog egymással implementációs szinten megjelenni.

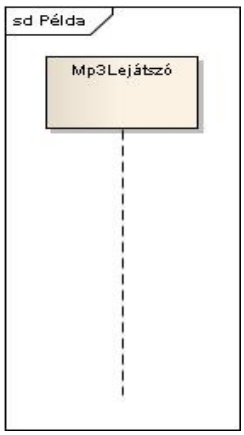
6.1. Szekvencia diagram

A rendszer működésének leírására szolgál a szekvenciadiagram. A probléma megoldása során az objektumok egymásnak üzeneteket küldenek. Az üzenetek időbeli sorrendjének szemléltetése gyakran megkönnyíti a probléma megoldásának megértését. Ez a feladata a

szekvencia diagramnak.
A szekvenciadiagram alapfogalmai, komponensei a következők:

- osztályszerep,
- osztályszerep életvonala,
- aktivációs életvonal,
- üzenet.

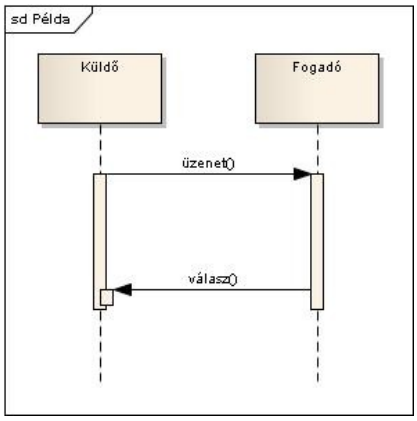
Az osztályok szerepét az osztályok közötti üzenetben megtestesítheti az osztály egy vagy több objektuma, amelyek az üzenetküldés szempontjából konform módon járnak el. Az osztályszerep megnyilvánulhat az osztályok egy halmazának megtestesítőjeként. Az életvonal az osztályszerep időben való létezését jelenti, ennek megszűnését a vonal végén x-el jelöljük. Az aktivációs életvonal az osztályszerepnek azt az állapotát jelöli, amelyben az osztályszerep megtestesítői műveletet hajtanak végre, és más objektumok vezérlése alatt állnak.
Egy objektum létrejöhet egy másik objektum létrehozó üzenetének a hatására, és megsemmisülhet, ha a másik objektum egy törlést jelentő üzenetet ad ki. Közben aktív módon viselkedhet. Az objektum aktivációjának egy speciális esete a rekurzív aktiváció, amikor egy objektum saját magát aktiválhatja. Ezen kívül az aktiváció lehet centrálisan vezérelt, amely esetben minden objektumot egy aktor objektum aktivizál.



6.17. ábra

Az objektumok aktivációs életvonalának alakulását az üzenetek szabályozzák. Ezek az üzenetek, ahogy az objektumokhoz viszonyulnak, osztályozhatóak a következő módon: egyszerű, szinkronizációs, randevú, aszinkron üzenet, s időhöz kötött várakozás. Az üzenet

egy példányának átadása általában egy esemény bekövetkezése. Az üzenet küldésének az a célja, hogy az objektum működésbe hozza a másik objektumot. Az üzenet azok között az objektumok között jöhet létre, amelyek az objektumdiagramban kapcsolatban állnak. Az üzenet az esemény egy példánya. Az üzenet küldése egy olyan akció, amelynek eredménye egy végrehajtható utasítás. Az üzenetnek van azonosítója (neve, szövege), lehet paramétere, sorszáma.



6.18. ábra

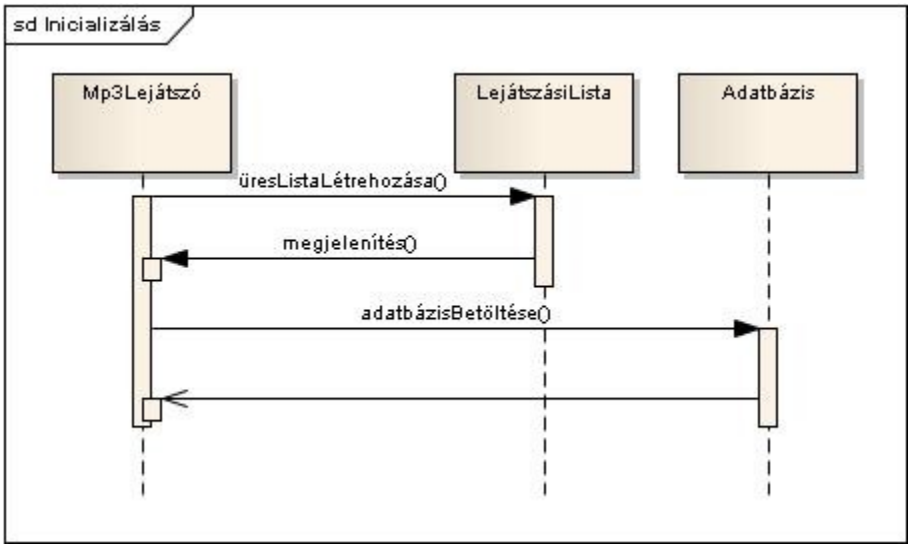
Üzenetek:

- *egyszerű üzenet*: egy aktív objektum üzenetet küld egy passzív objektumnak. Ez tulajdonképpen egy közönséges „call” utasítás. Jelölése telt fejű nyíllal történik.
- *szinkronizációs üzenet*: a küldő objektum elküldi az üzenetet, és a küldő blokkolt állapotba kerül, amíg a fogadó nem fogadta az üzenetet. A küldő várakozik, amíg a szinkronizációs feltétel nem teljesül.
- *időhöz kötött várakozás*: a küldő a megjelölt ideig helyezi magát várakozó állapotba.
- *randevú üzenet*: a fogadó várakozik arra, hogy a küldő üzenetet küldjön neki. A fogadó előbb várakozó állapotba helyezi magát a fogadáshoz.
- *aszinkron üzenet*: a küldő folyamat nem szakad meg, nem érdekli őt, hogy mikor kapta meg a fogadó üzenetet. Jelölése félfejű nyíllal történik.

6.2. Üzenetek az objektumok között

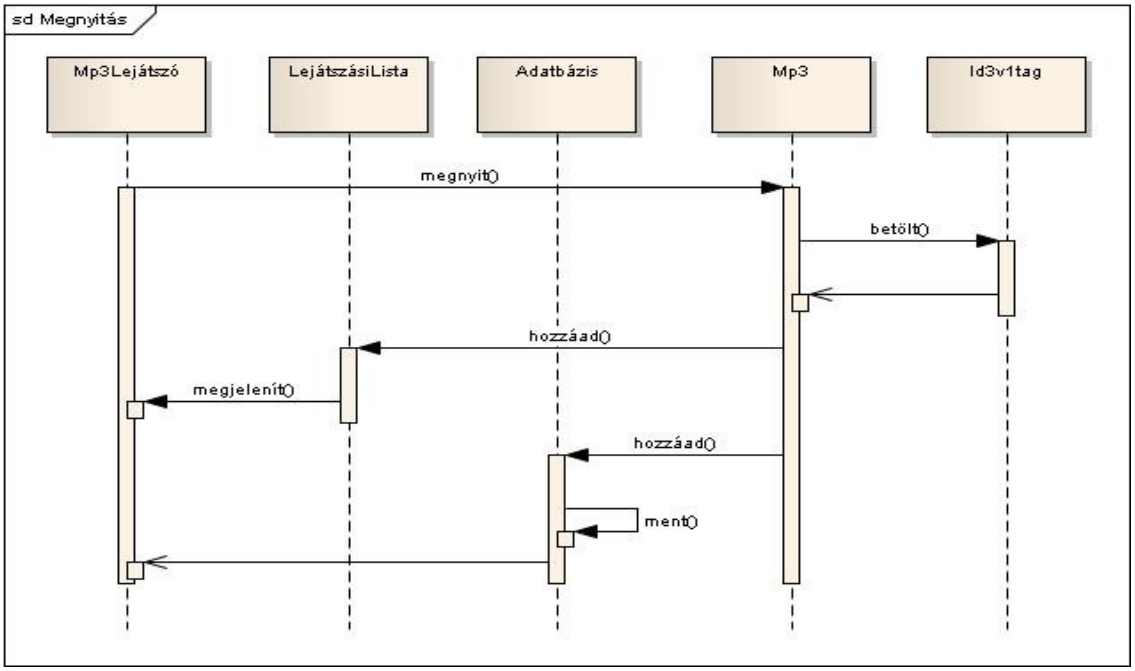
A szekvencia diagram megkonstruálásához össze kell gyűjtenünk az osztályainkat, ill. azok metódusait, valamint az üzleti logikánk egyes részfolyamatai szintén a segítségünkre lesznek. A példaprogramunkban a felhasználó tevékenységeit az Mp3lejátszón keresztül fogja lefuttatni. Ez abból adódik, hogy a jelenlegi rendszer tervezése során elhatároztuk, hogy az Mp3lejátszó tartalmazza majd a grafikai megjelenítést is, s a felhasználó ezt használva adja majd ki a parancsokat. Ez azt jelenti, hogy az üzleti folyamatban leírt folyamatok úgy fognak megvalósulni, hogy az Mp3lejátszó osztály hajtja őket végre, ill. ez az osztály fogja majd a többi osztályt létrehozni, valamint azoknak üzeneteket küldeni.
Az egyes részfolyamatok, úgymint, rendszer betöltése, fájlok megnyitása, lejátszás, lista kezelése, keresés, lista mentése, mind-mind lehetne egy diagramban ábrázolni, azonban ez már ennél a rendszernél is meglehetősen sok helyet foglal, így ezeket egyesével vizsgáljuk meg, s külön diagramon ábrázoljuk.

Ahogy látjuk egy külön folyamat jelenik meg, a rendszer betöltése, amely inkább az implementációs fázishoz tartozik. Ez egy olyan rész, amely minden rendszerben előfordul, egy általános rendszer követelmény, korábban nem írtuk le az üzleti logikánkban, mert annyira nem a rendszer funkcióihoz tartozik, azonban itt most megemlítjük, s megtervezzük. A rendszer betöltése úgy történik, hogy létrehoz egy új üres listát, valamint betölti az adatbázis.



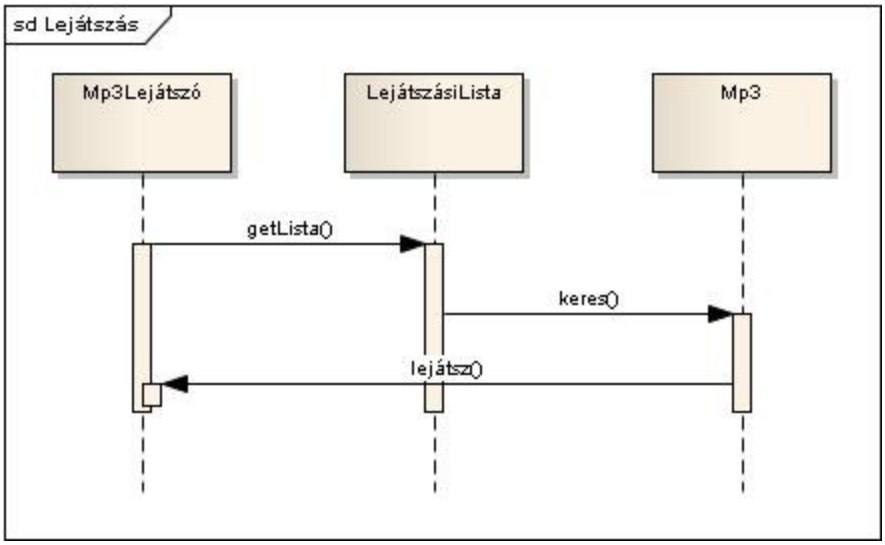
6.19. ábra

A fájlok megnyitása új Mp3 objektumok létrehozásával történik, melyek azután a lejátszási listába kerülnek. Ezek szerint az Mp3lejátszó által kiválasztott mp3-ak alapján létrehozunk új Mp3 objektumokat, majd azokat hozzáadjuk a lejátszási listánkhoz, valamint az adatbázishoz, ha még abban nem szerepelt és elmentjük az adatbázist. Az Id3v1tag-et az eredeti fájlból kiolvassuk.



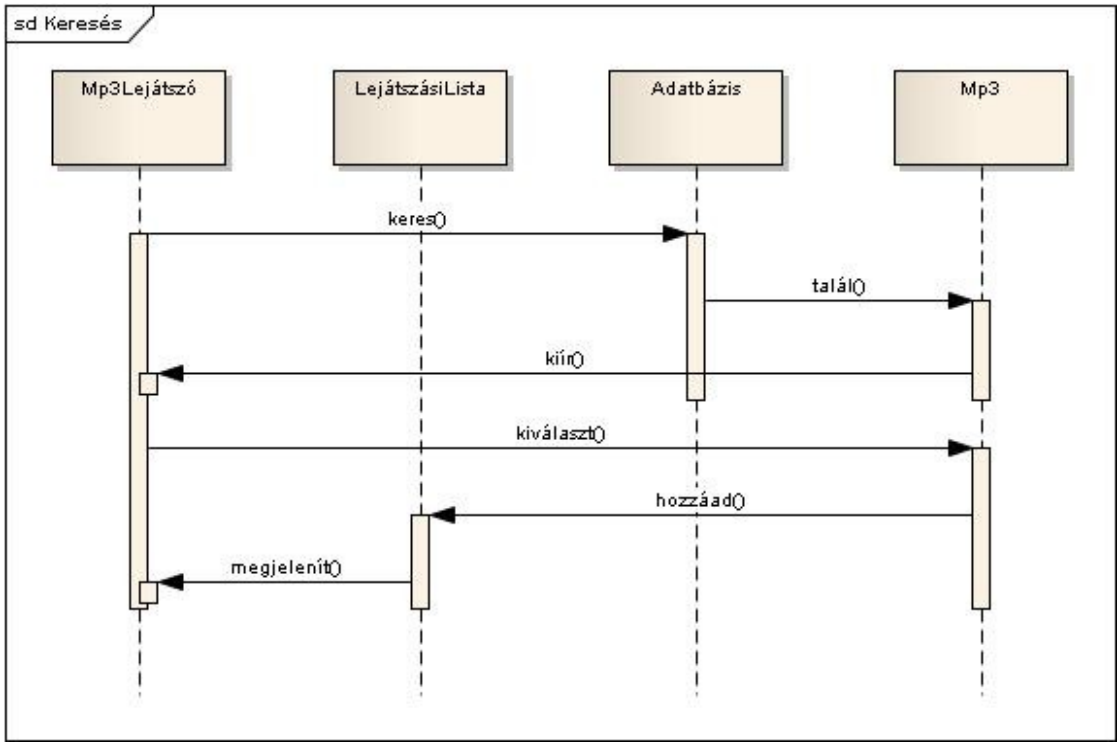
6.20. ábra

A lejátszás úgy történik, hogy mivel az Mp3Lejátszó grafikusan megjeleníti a Lejátszási listát, így abból kiválasztunk egy felvételt, s azt lejátssza a lejátszó.



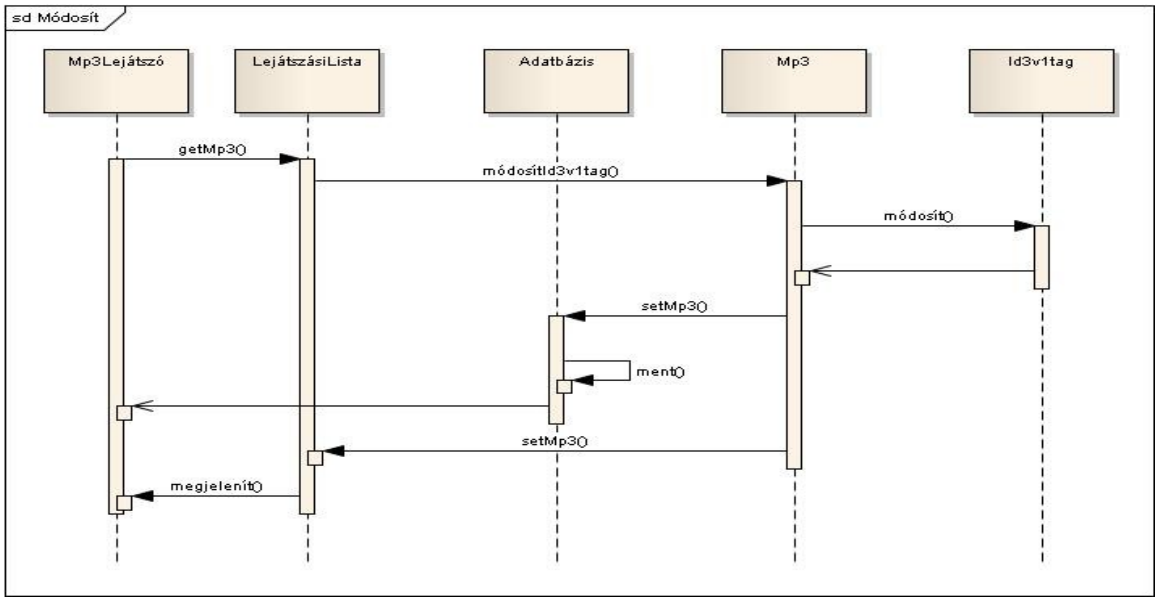
6.21. ábra

A keresés során a felület üzenetet küld a az adatbázisnak, abból megkeresi azokat a felvételeket, amelyek nevében szerepel a keresendő szó, s a találatokat kiírja a képernyőre. Ezt követően ezek közül választunk, majd azok hozzáadhatóak a listához, amit majd szintén az Mp3Lejátszó fog megjeleníteni.



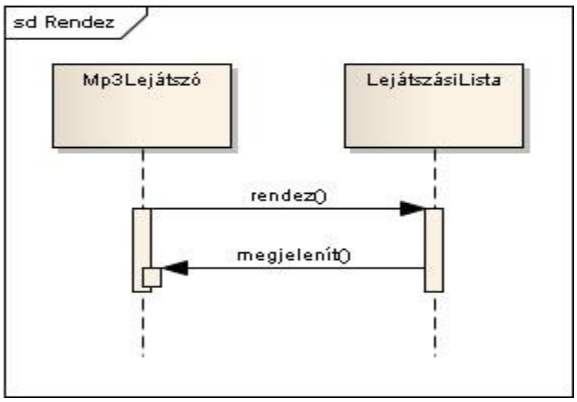
6.22. ábra

Az Id3v1tag módosítása során mind a lista, mind az adatbázisban levő mp3 fájlt reprezentáló objektumot elő kell keresni, s módosítani mindkettő Id3v1tag-jét. A tag-et az Mp3 fájlon keresztül érjük el. Miután megtörtént a módosítás, elmentjük az adatbázist.



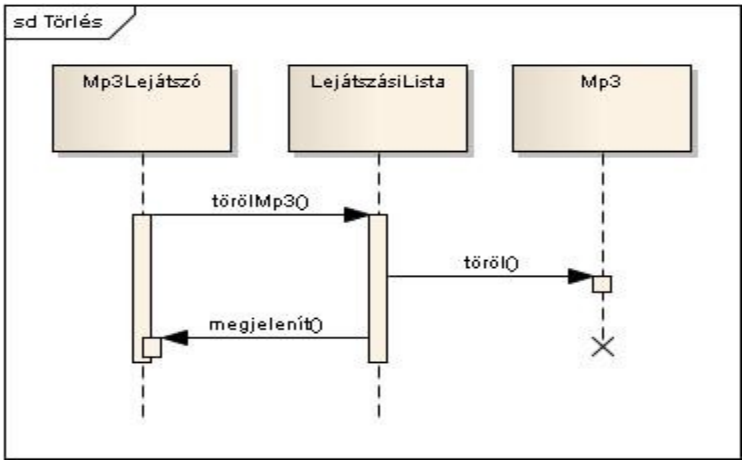
6.23.ábra

A rendezést a Lejátszó egy, a listának küldött rendez üzenettel fogja végrehajtani. Miután ez megtörtént a módosítások, pontosabban a számok új sorrendje frissül a lejátszó grafikus felületén.



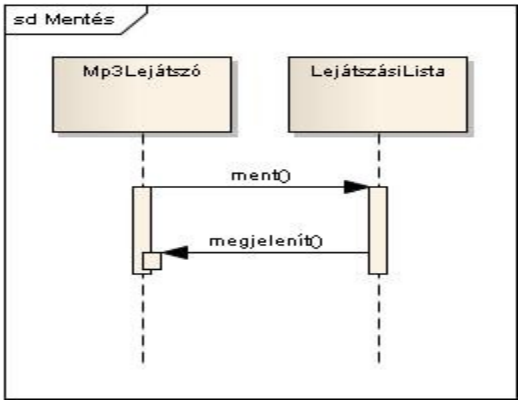
6.24.ábra

Az mp3 törlése az adott mp3 objektum megszüntetését jelenti, nem csak maga az objektum, hanem annak Id3v1tag-je is törlődik a rendszerből.



6.25.ábra

A mentés során a lejátszási lista kerül mentésre, ez is egy egyszerű metódushívás mint a korábbiakban.



6.26.ábra

Ezek az ábrázolások jelenleg meglehetősen elnagyoltak, nem igazán részletesek, ezeket tovább pontosíthatjuk részletezhetjük, ahogy a tervezés során egyre jobban átkerülünk az implementációs rétegbe, ahol már ténylegesen kiderül, hogy melyik objektum, hogy, mivel, kin keresztül fog kommunikálni. Ezzel a diagramtípussal elérkeztünk ahhoz a ponthoz, ahol a diagramból már kódot tudunk előállítani, megkezdhetjük a tényleges implementáció elkészítését.

6.3. Tovább a megvalósítás felé

Miután megalkottuk a statikus és a dinamikus modellt, azaz leírtuk az osztályhierarchiát, definiáltuk a felhasználói eseteket, az aktiviásokat, eseményeket, valamint az egyes objektumpéldányok közötti üzeneteket, a modellezés elérkezett a megvalósítási szinthez, a tényleges kód előállításához, implementáláshoz. Az elemzési és tervezési fázison kívül vannak implementációs szempont szerinti diagramok is, ezek egyike amelyről érdemes szót ejtenünk, a komponens diagram.

Az UML-ben a komponensek a fizikai és logikai rendszer azon moduláris részeit írják le, amelyek kifelé látható viselkedése jobban leírható, mint a megvalósításuk. A kifelé látható viselkedéseket interfészek halmaza reprezentálja. Két nézőpontot mutatnak, először is definiálják a rendszer részeinek külső arculatát, valamint megvalósítják a rendszer funkcionalitását.
A komponensdiagram a probléma megoldására szolgáló rendszer tulajdonságait implementációs szempont szerint fejezi ki.
A komponens alapfogalmai:

- komponens.
- reláció.

A komponens a rendszer egy fizikailag létező és kicserélhető része, feltéve, hogy a kicseréléshez az új komponens csatlakozási felületét a környezettel konform módon valósítjuk meg, azaz a környezethez az új komponens csatlakozási felületét hozzáillesztjük.
A komponensek közötti relációkat két csoportba sorolhatjuk:

- fejlesztés során fennálló reláció
- meghívási reláció

7. Összefoglalás

A legfontosabb diagramtípusokon, valamint egy kifejlesztendő mp3 lejátszó rendszerén keresztül bemutattuk az UML leíró nyelv használatát, gyakorlati alkalmazását az elemzés és a tervezés során, ahol ténylegesen szerephez jut, ill. amelyek a példaalkalmazás kifejlesztésében szerepet játszottak. Ezek a diagramok szerepelnek majd a szoftverhez való dokumentációkban, valamint nem csak ezek a diagramok, hanem az egyes változások is, ill. azt, hogy ezeket kik hajtották végre, s mikor. Azért választottuk a fent tárgyalt diagramtípusokat, mert leginkább ezek szerepelnek a szoftverkövetelmény specifikációban, és a szoftvertervezési dokumentumban amelyek egy kész tervet adnak a rendszer leimplementálásához.

Láttuk, hogy a dolgozat leginkább az elemzési és tervezési fázisra fókuszál, s remélhetőleg hasznos információkkal szolgál az olvasónak az UML modellező nyelvvel való modellezéshez. A példák és a bemutatott eszközök csak egy részét képezik az UML tárházának, rengeteg eszköz áll a rendelkezésre. Ezzel ellentétben viszont a nyelv még nem teljesen kiforrott, túl sok diagramot használ, melyek részben redundánsak, ezért bonyolulttá nehezen áttekinthetővé válhatnak. A szemantika nem definiált szigorúan, bizonyos értelembe vett rugalmassággal bír. Legnagyobb hátránya, hogy főleg kisebb projekteknél amennyi idő alatt egy modellt elkészítünk, annyi idő alatt végrehajtható maga az implementáció is.

A dolgozat elkészülésekor segítségével hozzájárult Pánovics János egyetemi tanársegéd akinek külön köszönetet szeretnék nyilvánítani a türelméért. Kollár Lajos szintén egyetemi tanársegéd óráin készült jegyzetek, valamint a számos könyv, prezentáció és dokumentum pedig nagyban hozzájárultak ahhoz, hogy a dolgozatban szereplő módszerekről, eszközökről az olvasó átfogó képet kapjon, s betekintést nyerjen az Unified Modeling Language gyakorlati alkalmazásába.

8. Irodalomjegyzék

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

Harald Störrle: UML 2, Panem Kiadó, 2007

Doug Rosenberg – Matt Stephens: Use Case Driven Object Modeling with UML: Theory and Practice, Apress, 2007

Scott W. Ambler: www.agilemodeling.com

Artiso Visual Case: www.visualcase.com