Eötvös Loránd Tudományegyetem
Informatikai Kar

Programozáselmélet és Szoftvertechnológiai Tanszék

Szoftvertechnológia

feladatgyűjtemény

Összeállították:

Dr. Sike Sándor

Giachetta Roberto

Budapest, 2015

1. Használati esetek

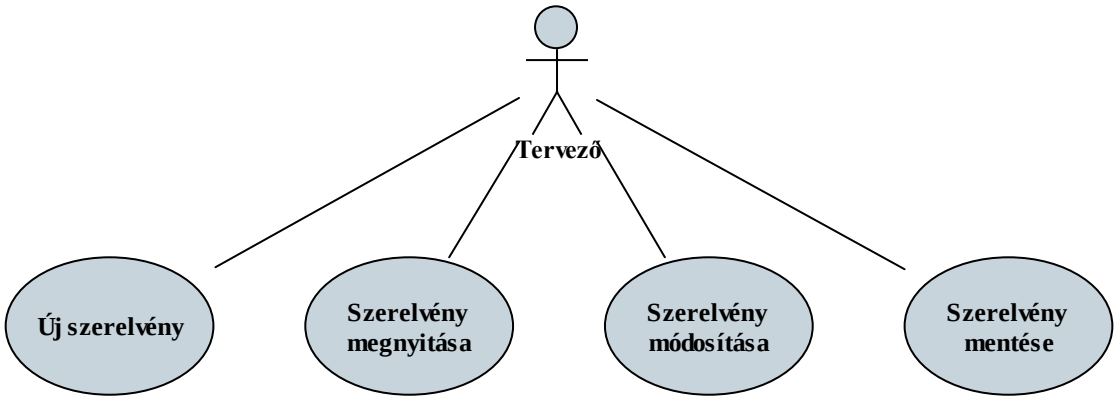
Az UML *használati esetek* (use case) egy szabványos, grafikus leírása a szoftver funkcionális követelményeinek. Azt ábrázolja, miként működik együtt a felhasználó a rendszerrel. Ehhez a diagramban funkciókat valamint aktorokat, illetve a köztük fennálló relációkat ábrázoljuk.

Mintafeladat

Egy ipari környezetben egy rendszer segítségével szerelvényeket terveznek, amelyek különböző alkatrészekből állnak össze. A rendszert tervezők használják, akik az egyes elemekből összeállítják a szerelvényeket. A tervező létrehozhat egy új szerelvényt, vagy betölthet egy létezőt. Ezután módosíthatja, majd elmentheti azt. A módosítás során a tervező alkatrészeket adhat a szerelvényhez, vagy vehet el belőle. Mindkét esetben ki kell választania a megfelelő alkatrészt. Minden módosítást követően a program automatikusan validálja a szerelvény konzisztenciáját. Az elmentett szerelvények műszaki ellenőrzését mérnökök végzik. Ők hagyják jóvá a szerelvényt, amennyiben nem találnak hibát.

1. *Szerelvények használati esetei*

A leírás alapján a szerelvényekkel kapcsolatosan tervezői szempontból négy fő funkciót különíthetünk el: új szerelvény létrehozása, szerelvény betöltése, szerelvény módosítása, valamint szerelvény mentése. Ezeket a funkciókat mind közvetlenül a tervező éri el a rendszerben, ő lesz a rendszer egyik felhasználója, így aktorként vesszük fel. Természetesen a funkciók és a tervező között használati (usage) relációt vehetünk fel (1. ábra).

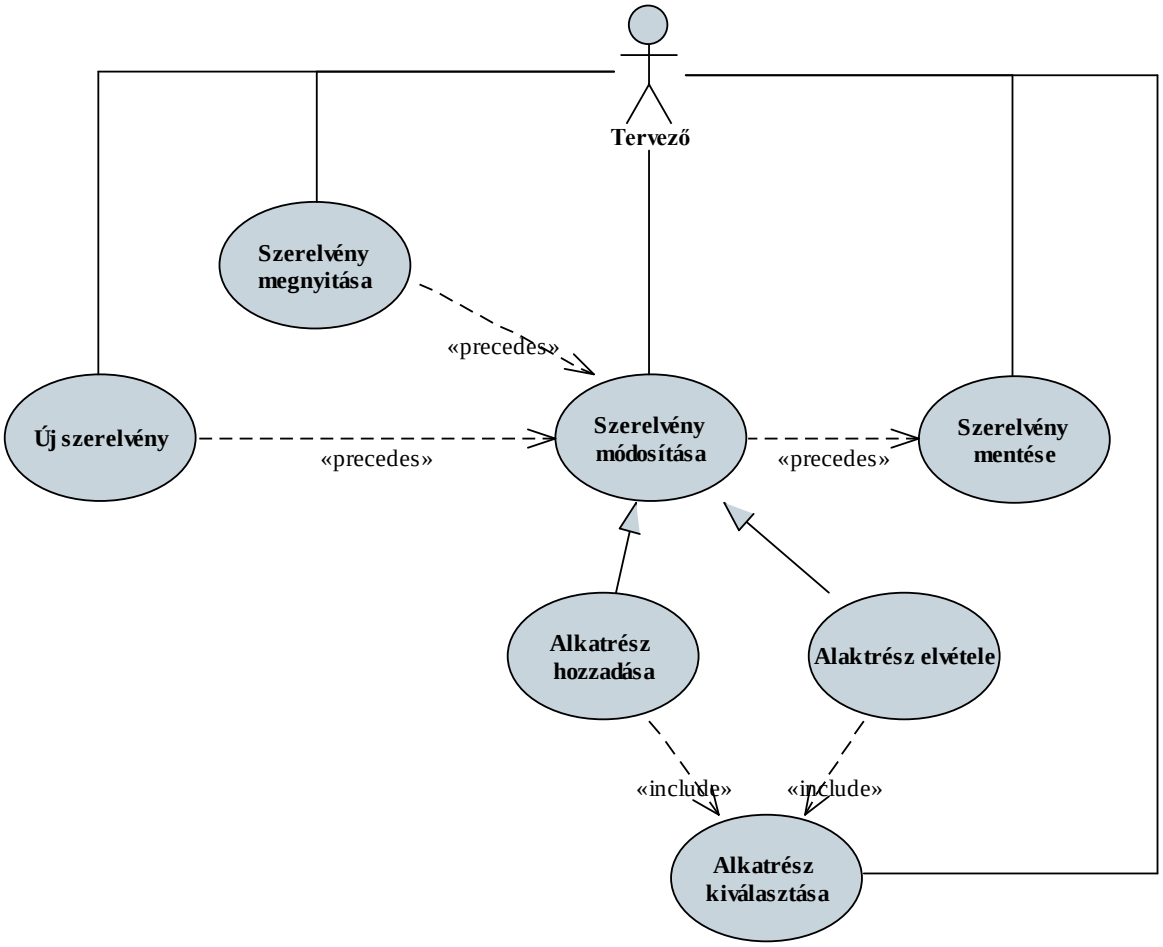


1. ábra: A terv alapvető használati esetei

2. *Használati esetek kapcsolatai, kifejtése*

A felvázolt funkciók nem érhetőek el azonnal a tervező számára. Ahhoz, hogy a szerelvény módosítását elvégezhesse, elsőként az „új szerelvény”, vagy a „szerelvény megnyitása” funkciókat kell használnia, így ezek megelőzik (precede) a módosítást. Hasonlóan a szerelvény mentése csak akkor van értelme, amennyiben a szerelvény módosult, így itt is egy megelőzési relációt vehetünk fel. A módosításnak két speciális

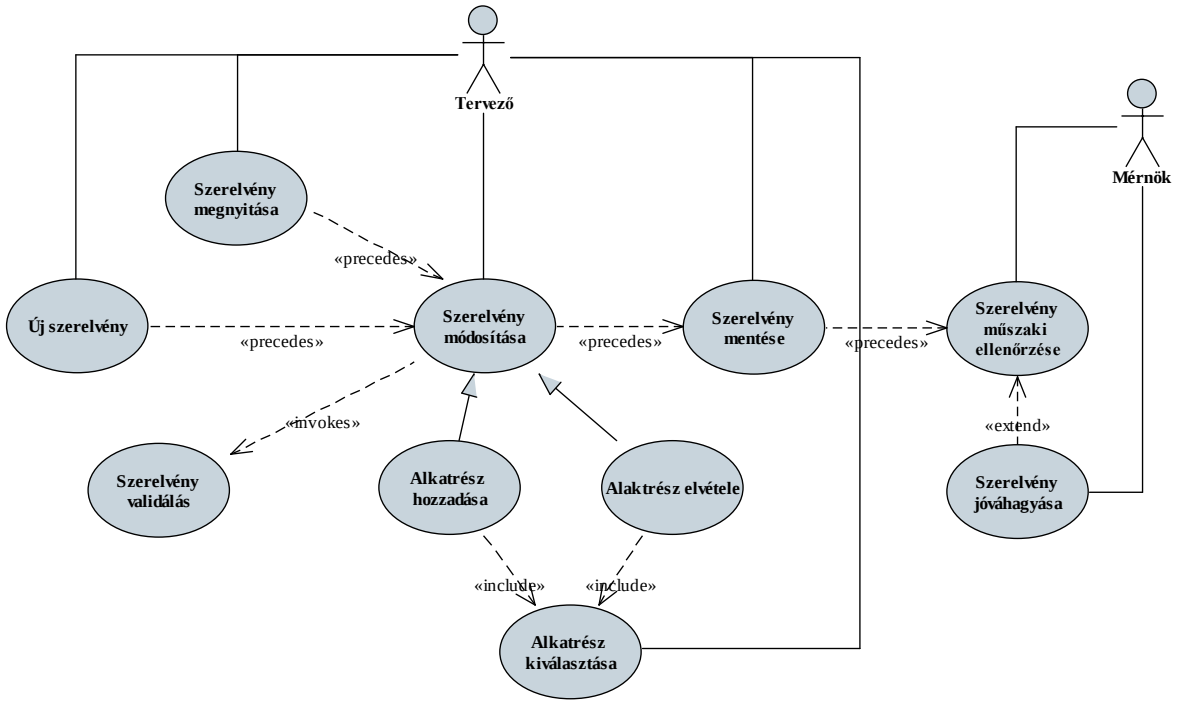
esetét ismerjük, alkatrész hozzáadását, illetve törlését, amiket általánosítás (generalization) segítségével jelölhetünk. (Természetesen a hozzáadást, valamint törlést felfoghatjuk úgy is, mint a módosítás egy-egy részfunkcióját, ebben az esetben szintén tartalmazást használhatunk.) Mindkét funkció részét képezi egy további, az alkatrész kiválasztása, amelyet tartalmazással (include) kapcsolhatunk hozzájuk, vagy az általánosabb módosításhoz. Ezt a funkciót szintén hozzákapcsolhatjuk a tervezőhöz (2. ábra).



2. ábra: A használati esetek kifejtése

3. *Szerelvények ellenőrzése*

A szerelvényhez köthetőek ellenőrzési funkciók is. Minden módosítást követően történik egy validálás, amit a rendszer végez. Ez automatikusan követi (invoke) a módosítást, és ez a funkció nincs kapcsolatban a felhasználóval. A műszaki ellenőrzést mérnökök végzik, a szerelvény mentését követően. Ez egy új felhasználói csoport, így új aktorral reprezentáljuk. A műszaki ellenőrzésen túl a jóváhagyás funkcióját is ők használják. Ez az eset többféleképpen is kapcsolatba hozható a műszaki ellenőrzéssel. Pl. tekinthetjük annak részeként, annak rákövetkezőjeként, illetve annak kiterjesztéseként (extend). Utóbbi alkalmazva állítjuk elő a végleges felhasználói eset ábrázolást (3. ábra).



3. ábra: Az ellenőrzés használati esetei

1.1. Feladat

Használati esetek:

Egy hotelben a vendégek foglalhatnak szobát a recepciós segítségével, aki bejelentkezteti őket érkezéskor, illetve kijelentkezteti őket távozáskor.

1.2. Feladat

Használati esetek:

Egy motoros játékban a játékos feladata, hogy minél tovább haladjon a motorjával, elkerülve az akadályokat. Ehhez először új játékot kezd, majd balra, illetve jobbra húzhat a motorjával, amíg akadálynak nem ütközik.

1.3. Feladat

Használati esetek:

A torpedó játékot két játékos játssza. Először mindkét játékos felpakolja hajóit a játéktáblára, majd felváltva lépkednek. A lépés lehet találat, vagy mellélövés. Kellő számú találat esetén az ellenfél elveszíti a hajóját.

1.4. Feladat

Használati esetek:

Egy alkalmazás célja a legmagasabb sziget megállapítása egy adott vonalban végzett mérések alapján. Ehhez először a felhasználó betölthet egy fájlt, vagy manuálisan megadhat magassági adatokat. Utóbbi esetben először megadja a pontok számát, majd magukat a pontokat. A bevitelt követően az alkalmazás kiírja a legmagasabb sziget sorszámát.

1.5. Feladat

Használati esetek:

Adott két szöveges állomány, amely egy-egy zárthelyi eredményeit tartalmazzák. A hallgató jegyét a két zárthelyi átlaga adja. Egy programmal szeretnénk megkeresni egy hallgató eredményét kikeresni. Ehhez előbb megadjuk a két fájl nevét (amennyiben helytelen, a felhasználónak újra meg kell adnia), majd megadhatjuk a keresett hallgató azonosítóját, amire a program megadja a hallgató eredményét.

1.6. Feladat

Használati esetek:

Az ATM automatákat ügyfelek használják. Először behelyezik a bankkártyájukat, majd megadják a PIN kódjukat. Ha hibás a PIN, akkor a gép újra bekéri, ha helyes, akkor az ügyfél megadhatja a levenni kívánt összeget. Az összeget a bank hitelesíti, az automata pedig kiadja a pénzt, és kinyomtatja a bizonylatot.

1.7. Feladat

Használati esetek:

Egy vizsgán a hallgatók egy tételgenerátor program segítségével „húzhatják” a tételüket. Ehhez először elindítják a generátort, majd leállítják. A generálás előtt az oktató beállítja a tételek számát, a teremben lévő hallgatók számát (amely megadja, a legutóbbi hány tétel nem adható ki), illetve tetszőlegesen ki/be kapcsolhatja a tételek kiadhatóságát.

1.8. Feladat

Használati esetek:

Egy mozi weblapján a felhasználóknak lehetőségünk van helyet foglalni, valamint jegyet vásárolni. Az előadást kiválasztva a weblap megadja az üres helyeket, amiből választhat, majd megadja nevét, illetve e-mail címét. Amennyiben vásárol, meg kell adnia bankkártya adatait is, amit a bank ellenőriz. Foglalás esetén egy azonosítót ad a rendszer, míg vásárlás esetén a kinyomtatható jegyeket.

1.9. Feladat

Használati esetek:

Egy blogon a szerkesztő létrehozhat bejegyzéseket, amelyekhez címkéket adhat, illetve kategóriába sorolhatja őket. A látogatók elolvashatják a blogbejegyzéseket, valamint kommentelhetik őket. Természetesen a szerkesztő is olvashatja a bejegyzéseit, illetve kommentelheti őket, ezen felül törölheti a bejegyzést, valamint bármelyik kommentet.

1.10. Feladat

Használati esetek:

Egy ételrendelő weblapra érkezve a felhasználó megtekintheti az étlapot, illetve itallapot, de rendelni csak regisztrációt, illetve bejelentkezést követően lehet. Regisztrációkor a rendszer küld egy aktivációs e-mailt. A bejelentkezett felhasználó böngészheti az étlapot, és a kosarába rakhatja a kiválasztott ételeket/italokat. Amennyiben pizzát választ, akkor még további feltételeket is megadhat hozzá. Rendelés közben bármikor megtekintheti a kosarát, ahol tudja törölni a tételeket, módosítani a

darabszámot, illetve megadhatja, hogy szeletelve kéri-e a pizzát. Végezetül leadhatja a rendelést.

1.11. Feladat

Használati esetek:

Egy időjárás alkalmazásban egy, vagy több város aktuális és jövőbeli időjárását tudjuk megtekinteni. Az alkalmazás lekér az eszköztől a tartózkodási helyünket, és amennyiben az elérhető, az adott város időjárását jeleníti meg. Amennyiben nem, az alapértelmezett városét. Ezen felül a felhasználó tetszőleges további várost adhat hozzá az alkalmazáshoz, amelyek között váltogathat az alkalmazásban. Az alkalmazásban az alapértelmezett város mellett beállíthatjuk, hogy Fahrenheitban, vagy Celsius-ban jelenítse meg a számokat.

1.12. Feladat

Használati esetek:

Egy taxi rendelő alkalmazásban az ügyfelek egy gombnyomásra rendelhetnek, csupán meg kell adnunk a nevünket és a telefonszámunkat. Az alkalmazás értesíti a legközelebbi szabad taxist, aki elfogadhatja a fuvart, vagy passzolhatja, ekkor a következő taxis kerül értesítésre. Amint egy taxis visszaigazolta a rendelést, az alkalmazásban megjelenik a pozíciója, távolsága, valamint a várható érkezési ideje. Ezeket az alkalmazás folyamatosan frissíti.

1.13. Feladat

Használati esetek:

Egy navigációs alkalmazás automatikusan felismeri a tartózkodási helyünket, ezért a felhasználónak csak a célállomást kell beállítania. Ha ez megtörtént, az alkalmazás kiszámítja a leggyorsabb útvonalat. A felhasználó erre elindítja a navigációt. Az alkalmazás folyamatosan figyeli a felhasználó tartózkodási helyét, és amennyiben az megváltozott, akkor frissíti a navigációt. Amennyiben a felhasználó letér a kijelölt útvonalról, akkor újratervezi azt.

1.14. Feladat

Használati esetek:

Egy grafikus rajzolóprogramban lehetőségünk van új rajz készítésére, fájlok betöltésére és mentésére (természetesen csak amennyiben már megnyitottunk, vagy betöltöttünk fájlt). Létrehozáskor meg kell adnunk a rajzvászon méretét (szélesség, hosszúság). A programban rajzolhatunk vonalat, téglalapot, valamint ellipszist, amelyeknek külön beállíthatjuk a kitöltését, valamint keretét. A kitöltésnél színt, és mintát, a keretnél pedig színt és szaggatottságot választhatunk ki. Lehetőségünk van az utolsó művelet visszavonására is (ha történt rajzolás), illetve újbóli alkalmazására, amennyiben már visszavontunk műveletet. Minden művelet után történik egy automatikus biztonsági mentés, így ha az alkalmazás összeomlik, a program automatikusan az utolsó biztonsági mentést tölti be.

2. Statikus ábrázolás

A statikus modellezés a rendszer szerkezetét, felépítését tárja fel az UML *osztálydiagram* (class diagram) és *objektumdiagram* (object diagram) segítségével. Összetettebb feladatok esetén a szerkezetet több csomagra, illetve komponensre is bonthatjuk, amelyek kapcsolatait szintén feltérképezhetjük csomagdiagram, komponensdiagram, illetve kihelyezési diagram segítségével.

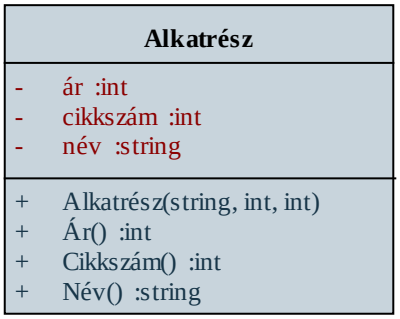
Mintafeladat

Egy ipari környezetben a gyártott szerelvények tetszőleges számú alkatrészekből állhatnak, illetve önmagában is tartalmazhat szerelvényeket, azaz hierarchikus felépítésűek (természetesen a legalsó szerkezeti szinten már csak alkatrészek állhatnak). Az alkatrészek tekintetében több típust különböztetünk meg, melyek más-más tulajdonságokkal rendelkeznek. Feladatunk az egyes szerelvények szerkezetének részletes ábrázolása, és annak megállapítása, hogy az alkatrész milyen költségen állhat elő, ha pusztán a benne lévő alkatrészek árait nézzük.

1. Alkatrész tervezése

Mivel minden alkatrésztípus külön árral rendelkezik, célszerű az egyes alkatrészeket külön objektumként kezelni, így mindegyikhez önálló attribútumokat társíthatunk. Ezek az objektumok bár többféle alkatrészt írnak le, ugyanolyan szerkezettel rendelkeznek, hiszen minden alkatrészről elég nyilvántartanunk annak nevét és cikkszámát az azonosításhoz, illetve az árát a későbbi számításokhoz. Annak érdekében, hogy kívülről ne lehessen módosítani, az adatokat elrejtjük, és lekérdező műveleteket biztosítunk. Tehát az alkatrészeket egy önálló osztályba soroljuk a következő mezőkkel: cikkszám (egész szám), név (szöveg), ár (egész szám)

Mivel már az egyes alkatrészek létrehozásakor meg kell adnunk paramétereit, egy konstruktor feladata lesz a paraméterek átadása a mezőknek. Értelemszerűen hiányos adatokkal alkatrészt nem hozhatunk létre. Az osztálynak megfelelő UML diagram a 4. ábrán látható.



4. ábra: Az alkatrész osztálydiagramja

Az alkatrész osztály C++ kódja:

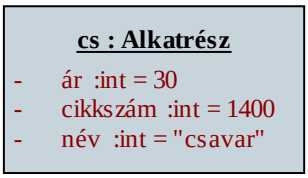
```
class Alkatresz {
public:
   Alkatresz(string n, int csz, int a) {
       nev = n;
       cikkszam = csz;
      ar = a;
   }
   string Nev() const { return nev; }
   int Cikkszam()const { return cikkszam; }
   int Ar()const { return ar; }

private:
   string nev;
   int cikkszam;
   int ar;
};
```

Ennek megfelelően vegyünk egy példányt az osztályból, például hozzunk létre egy 1400-as cikkszámú, 30 Ft-os csavart a következő C++ utasítással:

```
Alkatresz cs("csavar", 1400, 30);
```

Így létrejön egy objektum az osztályhoz, melynek objektumdiagramja látható az 5. ábrán. A szerelvények alkatrészekből állnak, és egy szerelvényben több ugyanolyan alkatrész is lehet. Bár ezek az alkatrészek ugyanazon tulajdonságokkal rendelkeznek, mégis több külön objektumként kell kezelnünk őket az összeszerelés során, így több példányt is létrehozunk azonos állapottal.

```
Alkatresz cs1("csavar", 1400, 30);
Alkatresz cs2("csavar", 1400, 30);
```



5. ábra: Egy alkatrész objektumdiagramja

Jól látható, hogy a két objektum paramétereiben teljesen megegyezik, és mivel ugyanazon osztály példányai, felépítésükben és műveleteikben is megegyeznek, azaz csupán nevük (azonosítójuk) különbözteti meg őket, de mivel két külön objektumként kezeljük, a memóriában két külön adatterületet fog elfoglalni. Ezen objektumok létrehozása során így a következő problémák merülnek fel:

- Ugyanilyen paraméterekkel rendelkező objektumból sok példányt hozhatunk létre, ami pazarláshoz, redundanciához vezetne, hiszen a memóriában többször tárolnánk el ugyanazokat az adatokat.

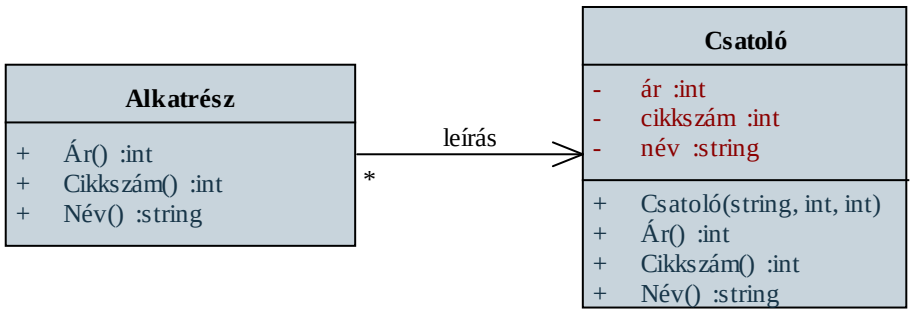
- Ha a későbbiekben módosítani akarnánk egy alkatrésztípus attribútumát (pl. emelkedett az ára), az összes meglévő objektumot módosítanunk kéne, ami rengeteg munkát jelent.
- Ha egy alkatrésztípusból az összes objektumot töröljük a rendszerből, azok attribútumai is elvesznek, pedig később lehet, hogy létre kívánunk hozni újabb példányokat.

2. *Alkatrészek leírása*

Célszerűbb lenne tehát az egyes alkatrésztípusok tulajdonságait külön eltárolni, egy másik objektumban, melyből csak egy létezne, és mely kapcsolatban állna az összes típusnak megfelelő alkatrészpéldánnyal. Ezek a leíró objektumok pusztán csatolt információkat tárolnak, így nevezzük őket csatolóknak. Ezzel a megközelítéssel mindhárom problémát megoldhatjuk:

- Az adatokat csak egy helyen tároljuk, megszűnik a redundancia.
- Az esetleges módosítást csupán egy helyen kell elvégeznünk.
- A csatoló a konkrét alkatrészektől függetlenül létezhet, így az adatok bármikor megtalálhatóak a rendszerben.

A módosított tervbe bekerül egy második osztály is, hiszen a csatolóknak is önálló struktúrával kell rendelkezniük, ugyanakkor minden alkatrésznek kapcsolatban kell állnia egy csatoló objektummal, mivel az alkatrészek ezek után önállóan már nem tartalmaznak adatokat. A két objektum így relációban lesz egymással, az alkatrész leírója lesz a csatoló. Az alkatrész tulajdonságait továbbra is az alkatrésztől kérdezzük le, amely továbbítja a lekérdezést a hozzákapcsolt csatolóhoz. A csatolón keresztül viszont nincs módunk a hozzákapcsolt alkatrészek eléréséhez, vagyis ez egy aszimmetrikus kapcsolat, amelynek osztálydiagramja látható a 6. ábrán.



6. ábra: Alkatrész és Csatoló osztálydiagramja

Ezek alapján már implementálhatjuk a két osztályt. Nyilván a csatoló felépítése megegyezik az eddigi alkatrészével, hiszen ugyanazon adatokkal kell rendelkeznie, és csak az ezeket kiolvasó műveleteket kell implementálnunk. Az *Alkatrész* elveszti eddigi adattagjait, pusztán annyi információt kell tárolnia, hogy elérhesse a hozzá tartozó csatolót, amit egy *Csatoló* objektumra történő hivatkozással teszünk

lehetővé. Természetesen ezt már az alkatrész objektum létrehozásakor meg kell adnunk, így a konstruktornak csak a csatolóra történő hivatkozást kell továbbadnia.

Emellett az *Alkatrész* osztály szerkezete nem változik, így a külső viselkedése megegyezik a korábbival, azaz ezen osztály objektumaihoz való hozzáférések a programban változatlanok maradhatnak.

```
class Csatolo {
public:
   Csatolo(string n, int csz, int a);
   string Nev() const { return nev ; }
   int Cikkszam() const { return cikkszam; }
   int Ar() const { return ar; }

private:
   string nev;
   int cikkszam;
   int ar;
};

class Alkatresz {
public:
   Alkatresz (const Csatolo& cs) : csat(cs) { }
   string Nev() const { return csat.Nev(); }
   int Cikkszam() const { return csat.Cikkszam(); }
   int Ar() const { return csat.Ar(); }

private:
   const Csatolo& csat;
};
```

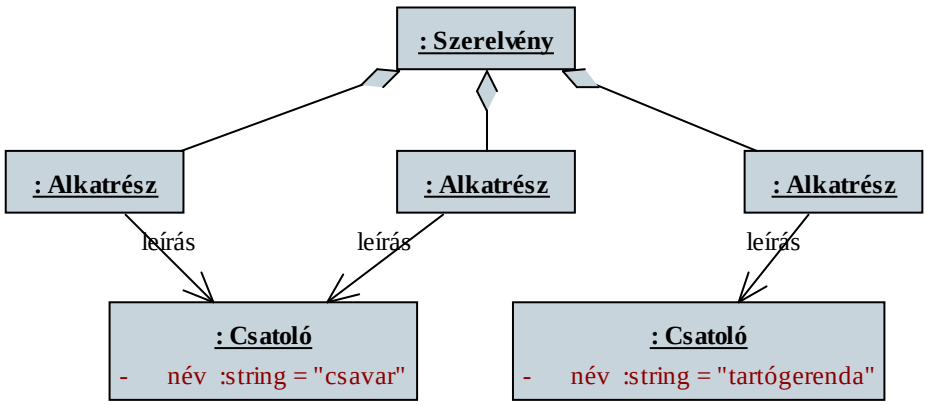
Ennek folyományaként az alkatrészek létrehozása több lépésből fog állni. Először létre kell hoznunk a megfelelő csatoló objektumot, majd ez után hozhatjuk létre az adott típusú alkatrészt. Amint egy adott alkatrész típushoz létrehoztuk a csatolót a programban, a többi ugyanilyen típusú alkatrészhez már nem kell újabb csatolót létrehoznunk. A fent látott konstrukció tehát a következőképpen valósítható meg:

```
Csatolo csatolo1("csavar", 1400, 30);
Alkatresz cs1(csatolo1);
Alkatresz cs2(csatolo1);
```

Az *Alkatresz* osztály elsődleges feladata továbbra is az, hogy adatokat szolgáltasson magáról, amit üzenetküldéssel valósít meg. Az üzenetküldés során a program valamely más komponense, egy *kliens* objektum lekérdezheti például az alkatrész árát az *Ár* művelettel. Az alkatrész megvalósításában fogadja az ár műveletet, ám azt rögtön tovább is adja a hozzá tartozó csatolónak, hiszen saját maga nem rendelkezik az adattal, a csatoló objektum viszont már képes visszaadni az adatot, amely így eljut a *kliens*hez.

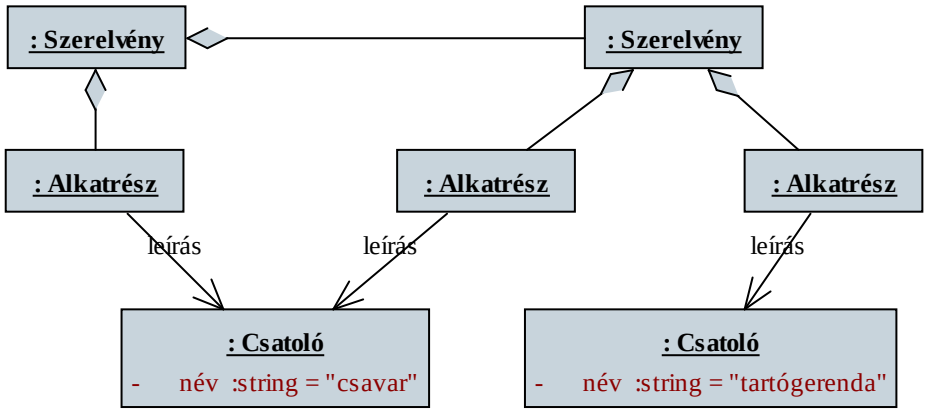
3. Szerelvények tervezése

A szoftvernek szerelvényekkel is dolgoznia kell, így az alkatrészek adatai mellett az egyes szerelvények pontos felépítését is el kell tárolnunk. Szükségünk lesz egy új osztályra, mely a szerelvény objektumok működését megvalósítja. A *Szerelvény* osztály megvalósítását többféleképpen is elképzelhetjük, például a *Szerelvény* objektumok olyan adatszerkezetekkel rendelkezhetnek, amelyek több más objektumra is képesek hivatkozni, így míg egy alkatrész csak egy csatolóval állt kapcsolatban, a szerelvények több más objektummal is kapcsolatban lehetnek, legyenek azok alkatrészek, vagy más szerelvények (hiszen megengedtük, hogy a szerelvények szerelvényekből is felépülhessenek). A kapcsolat a szerelvény esetében tartalmazást jelent, amit aggregációval írhatunk le, hiszen a szerelvényben lévő alkatrészek kiszerelhetőek, illetve átszerelhetőek más alkatrészekbe. A 7. ábrán látható példa egy egyszerű szerelvényre, amely egy tartógerendából és két csavarból áll.



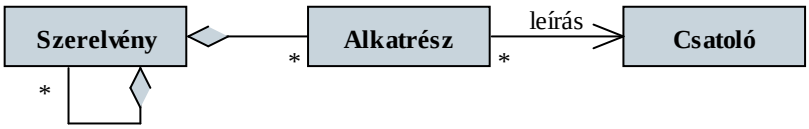
7. ábra: Egy szerelvény objektumdiagramja

Mivel szerelvény tartalmazhat szerelvényt is, megtehetjük, hogy előbb két alkatrészt szerelünk össze, majd az így kapott szerelvényt és a harmadik alkatrészt összeszerelve megkapjuk az előbbi szerelvényt, ám az szerkezetileg mégis eltérő lesz (8. ábra).



8. ábra: Egy szerelvény alternatív felépítése

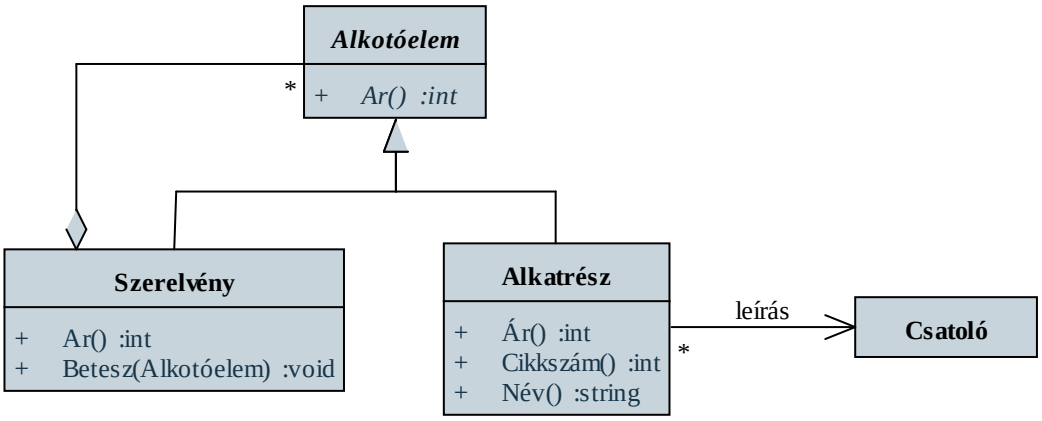
A hierarchikus szerkezet tehát lehetővé teszi a többszintű összeépítést is, ami a modellezés szempontjából annyit jelent, hogy a ”tartalmaz” kapcsolat fennállhat két szerelvény között, valamint egy alkatrész és egy szerelvény között. Ez jól ábrázolható az osztálydiagramon is (9. ábra).



9. ábra: A szerelvények osztálydiagramja

Ha ténylegesen így implementálnánk ezt az osztályt, az azt jelentené, hogy két külön megoldás, illetve adatszerkezet kellene a szerelvényekkel és az alkatrészekkel történő összekapcsoláshoz, ráadásul ezzel az UML diagramunk is elveszítené típusosságát, hiszen ugyanazt a kapcsolatot két különböző osztállyal tartanánk fent. A gyakorlatban ez annyit tesz, hogyha például módosítani szeretnénk egy szerelvény összeszerelését, akkor egy alkatrész helyére nem rakhatunk szerelvényt, és fordítva, így ezzel jelentősen korlátoznánk lehetőségeinket.

A megoldás itt egy olyan általános osztály létrehozása, melyre alkalmazhatjuk a tartalmazás asszociációt, ugyanakkor az általános osztály speciális esetei lehetnek a konkrét osztályok. Így vegyük be a szerkezetbe az *Alkotóelem* osztályt, melynek specializációi lesznek az *Alkatrész* és a *Szerelvény*. Az *Alkotóelem* egy absztrakt osztály, amely lehetőséget biztosít az ár lekérdezésére, de nem valósítja meg a műveletet, ezért azt leszármazottainak kell (10. ábra).



10. ábra: A szerelvények osztálydiagramja általánosítással

A specializáció implementálásához öröklődést alkalmazunk, ahol az ősosztály szerepét az *Alkotóelem* veszi át, és ebből származtatjuk két speciális osztályunkat. Alkotóelemeket sohasem fogunk betenni szerelvényekbe, hanem csak a konkrét alkatrészt, vagy szerelvényt.

Az ármeghatározás esetében egy kliens egy szerelvénytől kérdezi le az ő árát. A szerelvény ezt az üzenetet továbbküldi az őt alkotó alkatrészeknek és szerelvényeknek. Nyilván a szerelvény alkotóelemek is hasonlóan járnak el, míg az alkatrészek csatolójuktól kérdezik le a kívánt adatot, majd a kapott adatokat az egyes *Szerelvény* objektumok összegzik, és visszaadják.

Azonban egy szerelvény nem tudhatja, hogy milyen alkotóelemek tartoznak hozzá, hiszen az csak absztrakt *Alkotóelem* objektumokat lát, ezért már az absztrakt osztálynak is tartalmaznia kell a megfelelő művelet felületét, ám megvalósítását nem, hiszen mint azt láttuk, a két alkotóelem fajta teljesen máshogy valósítja meg például az *Ár* lekérdezést.

Az alkotóelemek hozzárendelését az egyes szerelvényekhez mutató segítségével valósítjuk meg, a különbség csak annyi, hogy mivel a szerelvény tetszőlegesen sok mutatót tartalmazhat, a mutatókat egy tömbben kell tárolnunk.

```
class Alkotoelem {
public:
   virtual int Ar() const = 0;
protected:
   Alkotoelem() {}
};

class Szerelveny : public Alkotoelem {
public:
   int Ar() const {
      int sum = 0;
      for (int i = 0; i < elemek.size(); i++ ){
         sum = sum + elemek[i]->Ar();
      }
      return sum;
   };

   void Betesz(Alkotoelem *a) { elemek.push_back(a); }

private:
   vector<Alkotoelem*> elemek;
};

class Alkatresz {
public:
   Alkatresz (const Csatolo& cs) : csat(cs) { }
   string Nev() const { return csat.Nev(); }
   int Cikkszam() const { return csat.Cikkszam(); }
   int Ar() const { return csat.Ar(); }

private:
   const Csatolo& csat;
};
```

2.1. Feladat

Osztálydiagram:

Egy lakóház legalább két szintből áll, szintenként legalább egy lakás található. A szinteket lépcsőházak, illetve liftek kötik össze. Minden szinthez legalább egy lépcsőház kapcsolódik. Minden lakásnak legfeljebb két szomszédja lehet.

Objektumdiagram:

A ház két szintből áll, az első szinten van az 1 és 2 számú lakás, amelyek szomszédok, a második szinten a 3, 4 és 5 számú lakás, amelyek közül a négyes szomszédos a másik kettővel. A házban egy lépcsőház található.

2.2. Feladat

Osztálydiagram:

Egy képkezelő programban a kép képpontokból (pixelekből) áll. Minden pixel a piros, kék és zöld színek 0..255 közötti értékeit tartalmazza. A képpontot lehet átlagolni, invertálni, és világosítani (egy egész értékkel). Ezek a műveletek a teljes képre is érvényesek, emellett a kép tükrözhető vízszintesen, függőlegesen, illetve mindkét irányba, elforgatható tetszőleges szöggel, betölthető és elmenthető a fájlnév megadásával. A program a változtatások visszavonása érdekében minden művelet alkalmazásakor egy új képet hoz létre, amelyet egy verem segítségével tárol el, így a visszavonás műveletnél egyszerűen törli a verem tetőelemét.

2.3. Feladat

Osztálydiagram:

A termelősor termékeket gyárt, amelyek adott alkatrészlistával rendelkeznek. A termelősoron ipari robotok dolgoznak (a sor méretétől függően 1-10), amelyek sorban megkapják a hozzájuk érkező, sorszámmal rendelkező alkatrészeket, és beépítik a termékbe.

2.4. Feladat

Osztálydiagram:

Az ATM automatákat ügyfelek használják. Az ügyfelek bankkártyákkal rendelkeznek, amikhez tartozik egy PIN kód, valamint egy bankszámla. A bankszámlának egyenlege van, ami mindig pozitív kell, hogy legyen. Az ügyfelek sorban vehetnek fel adott összeget az automatából a bankkártyájuk, valamint a kódjuk megadásával, ha a kód megegyezik a kártya kódjával, akkor az automata kiadja az összeget, feltéve, hogy az összeget levonva az egyenlegből az továbbra is pozitív marad. Ennek a megállapításához az automata egy központból a kártya adatainak megadásával visszakapja az ügyfél egyenlegét.

2.5. Feladat

Osztálydiagram:

A személyekről ismerjük a nevüket, a személyi igazolvány számukat és a címüket. Két személy között egy lehetséges viszony a házasság. Egy személynek legfeljebb egy házastársa lehet. A házasság jellemzője a házasságkötés helye és ideje. A házasságkötésen pontosan két személy tanúskodik. A házasságkötés során a házasulandók nászajándékokat kapnak, amelyeket legalább egy személy ad. A nászajándék jellemzője a név.

2.6. Feladat

Osztálydiagram:

Egy számítógépes fájlrendszerben a fájlokat könyvtárakba szervezzük. Minden könyvtár tetszőleges számú fájlt vagy könyvtárat tartalmazhat. A fájlrendszerben a fájlok lehetnek közvetlen a fájlrendszerhez kötve (gyökér), vagy valamelyik könyvtárban is elhelyezkedhetnek.

2.7. Feladat

Osztálydiagram:

Az alkalmazottakról nyilvántartjuk a nevüket, a címüket és a társadalombiztosítási azonosító jelet. Az alkalmazottak közül a főnök irányítja a beosztottakat. Az irányítás egy feladat határidőre történő megoldását foglalja magában.

Objektumdiagram:

Pál főnökként irányítja János és Jakab munkáját a tervezés feladatában, amelynek határideje 2003 november vége.

2.8. Feladat

Osztálydiagram:

Egy kertet egy kertész gondoz. A kert parcellákból áll, minden parcellába egyféle növény ültethető. A növények lehetnek haszonnövények, mint burgonya, borsó, paprika; vagy virágok, mint rózsa, szegfű, tulipán.

2.9. Feladat

Osztálydiagram:

Egy könyv oldalakból áll, amelyek lehetnek fejezetcímek, fejezetoldalak, illetve tartalom. A könyvet több szerző, vagyis író és szerkesztő készítheti, ezek egyike a vezető szerző. A könyvből meghatározott példányszámot adhat ki egy kiadó. A példányszámokat a kiadó több nyomdából rendeli meg. Minden nyomda rendelkezik egy aktuális kapacitással, és ezek együttesen ki kell, hogy adják a példányszámot.

2.10. Feladat

Osztálydiagram:

Egy programban különböző geometriai alakzatokat kezelünk, ezek lehetnek pont, vonal, illetve sokszög (poligon). A pont két valós szám, a vonal két pont, míg a sokszög tetszőlegesen sok, de legalább 3 pont segítségével adható meg. Minden geometriai alakzatnak le tudjuk kérdezni a területét, illetve kerületét, amelyet a különböző esetekben másként számolunk ki. Ezen kívül minden alakzatot eltolhatunk, illetve nagyíthatunk. A nagyításhoz elég egy valós szorzó tényezőt megadnunk, az eltolást viszont egy vektor segítségével végezzük, amelynek a ponthoz hasonlóan két valós számmal adhatunk meg, és lekérdezhetjük a hosszát.

2.11. Feladat

Osztálydiagram:

Egy személyszállító vonat egy mozdonyból és legalább egy kocsiból áll. A kocsikat a mozdony után adott sorrend szerint kapcsolják össze. A vonatot különböző típusú kocsikból állíthatják össze. A lehetséges típusok: első osztályú, másodosztályú, posta, étkező, háló. Egy mozdony, illetve kocsi egy időben csak egy vonathoz tartozhat.

2.12. Feladat

Osztálydiagram:

a) A kórházban legalább egy alkalmazott dolgozik, aki lehet orvos, ápoló vagy a személyzethez tartozhat. A kórházban betegek vannak, akiket orvosok kezelnek és

ápolók ápolnak. Egy beteget pontosan egy orvos kezel, és tetszőleges számú ápoló ápol. Egy orvos tetszőleges számú beteget kezelhet, egy ápoló tetszőleges számú beteget ápolhat.

b) Vegyük figyelembe azt is, hogy egy alkalmazott is lehet beteg, és a betegséggel nem szűnik meg az alkalmazotti viszonya.

Objektumdiagram:
Péter és Pál betegek, akik a kórházban fekszenek. Mindkettejüket Ottó, a kórház orvosa kezeli, Éva és Judit ápolók ápolják.

2.13. Feladat

Osztálydiagram:
Egy multinacionális cég országonként alkalmaz dolgozókat, akik lehetnek eladók, sofőrök, munkások, illetve menedzserek. Ugyanúgy országonként adott településekben létesítményei vannak, amelyek lehetnek irodák, gyárak, raktárak, illetve boltok. A létesítményeket az irodából irányítják a menedzserek. A gyárak gyártják a terméket, amelyet az adott kapacitású raktárak tárolnak, és amelyeket a boltokban adnak el.

2.14. Feladat

Osztálydiagram:
A koordinátarendszerben geometriai alakzatok találhatóak, amelyeket koordinátatömbök segítségével adhatunk meg. Ezek lehetnek pontok, szakaszok, háromszögek és négyszögek. Koordinátarendszerünk lehet 2, illetve 3 dimenziós, és ennek következtében az alakzatokat leíró koordináták is rendelkezhetnek 2, illetve 3 értékkel. A szakaszoknak lekérdezhetjük a hosszát, a háromszögeknek és négyszögeknek pedig területtel, illetve kerülettel rendelkeznek.

2.15. Feladat

Osztálydiagram:
Egy villamost kocsiszámmal rendelkező villamoskocsik alkotnak, amelyek lehetnek vezetőfülkések, vagy fülke nélküliek. A villamosnak ismert a kapacitása (mennyi ember fér bele), illetve az aktuálisan szállított utasok száma. Egy villamosnak legalább két fülkés kocsiból kell állnia, ezen felül tetszőleges kocsikat hozzá lehet csatlakoztatni. Az utasok megállókban várakoznak, ahol a villamosok a menetrend szerint sorban megállnak. Az utasok lehetnek kezdők, gyakorlottak, illetve profik. A kezdő utas csak akkor száll fel a villamosra, ha még van hely (az utasok száma kisebb, mint a kapacitás). A gyakorlott utas akkor is felszáll, ha nincs hely. A profi utas leráncigál a villamosról annyi embert, hogy legyen hely, majd felszáll.

Objektumdiagram:
A 6-os villamos a 152-es, a 155-ös, valamint a 106-os fülkés kocsiból áll. Sánta Rezső gyakorlott utas az Nyugati pályaudvarnál száll fel, Gröné Sára profi utas pedig a Boráros téren akarna felszállni, de mivel a villamos megtelt, előbb leráncigálja Sánta Rezsőt.

2.16. Feladat

Osztálydiagram:
Egy szállítmányozási vállalatnál a teherautók csak egyféle árut szállíthatnak, ami lehet papír vagy festék. A mennyiségen kívül tudnunk kell, hogy milyen minőségű papírt, illetve hány különböző színű festéket szállítanak. A vállalathoz tartoznak raktárak, illetve áruházak, a

teherautók ezekbe szállítják az árut. (Ügyeljünk arra, hogy a szállítás célja lehet áruház és raktár is.)

Objektumdiagram:

A HungaroSped vállalat három teherautóval rendelkezik, az első 30 tonna kiváló minőségű papírt szállít a 105-as áruházba, a második 20 tonna közepes minőségű papírt szállít a 336-os áruházba, míg a harmadik szintén a 336-osba szállít 5 tonna festéket, 12 féle színben. Ezen kívül a vállalat rendelkezik egy raktárral is.

2.17. Feladat

Osztálydiagram:

Egy űrhajó lehet kereskedő, illetve katonai. A kereskedők maximum egy, vagy két hajtóműből, maximum két raktérből, illetve egy irányítófülkéből áll, továbbá minden űrhajó tulajdonsága a típusa, az azonosítója, illetve a tulajdonos neve. Az űrhajót pilóták vezetik, akik pénzzel rendelkeznek. A raktérben árukat szállítanak, minden árunak megvan a tömege, az ára illetve a mérete. A raktérbe csak olyan áruk tehetőek, amelyek mérete kisebb a raktér méreténél, illetve a tárolt áruk össztömege nem haladja meg a raktér összeterhelhetőségét. A hajtómű - amelyet a pilótafülkéből irányít a pilóta - adott sebességre képes adott fogyasztás mellett. A kereskedő hajók kereskedhetnek egymással, feltéve, hogy a rakterükben van valami, és van a pilótáknál elég pénz, hogy legalább egy árut tudjanak venni a másik űrhajótól.

Objektumdiagram:

Az FE3128-as azonosítójú, Challenger típusú hajó tulajdonosa és pilótája Neil Armstrong, jelenleg 2000 Eurot birtokol. Ez a két hajtóműves, egy rakteres, 1600 m^3-es kapacitású hajó jelenleg 600 m^3 árut szállít, hatszoros fénysebességre képes 8 egység üzemanyag fogyasztása mellett, és jelenleg kereskedik az XT764-es azonosítójú T típusú hajóval, amelyet Edwin Aldrin vezet, aki 6000 Euroval rendelkezik. Ez a hajó két 800 m^3-es kapacitású, 800 kg-os terhelhetőségű raktárral rendelkezik, és egy hajtóművel, mely ötszörös fénysebességre képes, és 6 egység üzemanyagot fogyaszt.

2.18. Feladat

Osztálydiagram:

Grafikus felületű ablakoknál (Window) el kell tárolnunk az elhelyezkedést, a méretet, illetve a feliratot. Az ablak felületén tetszőleges számban lehetnek gombok (Button), illetve címkék (Label), mindegyik egy elhelyezkedéssel és egy szöveggel rendelkezik, ami megjelenik a felületén. A gomboknál ezen felül le lehet kérdezni az állapotukat is. A gombok speciálisan lehetnek nyomógombok (PushButton), kijelölőmezők (Checkbox) és rádiógombok (RadioButton), utóbbi kettőnél el kell tárolnunk, hogy ki vannak-e jelölve. Rádiógombokat csoportosíthatunk is (RadioButtonGroup), egy csoportban legalább kettő gombnak kell lennie, és persze ilyen csoportokból is tetszőleges számú lehet az ablakban. Az, hogy a gombok milyen állapottal rendelkeznek kezdetben az ablak betöltő eljárása mondja meg, míg az ablak bezárásakor a kimentő eljárással tároljuk el őket.

2.19. Feladat

Osztálydiagram:

Egy mozaikkirakó programban a mozaikot 4*4-es, képeket tartalmazó mátrix alkotja. A képek RGB (piros, zöld, kék) képpontokból állnak, amelyeket három egész paraméterrel beállíthatunk, és a pont koordinátája alapján lekérdezhetünk. A mozaikot a fájlnév alapján hozhatjuk létre, összekeverhetjük a képrészeit, ellenőrizhetjük, hogy kiraktuk-e, valamint megcserélhetünk két

képrészt a sorszám alapján. Ehhez a mozaikban eltároljuk, mi a képrészek helyes aktuális sorrendje, valamint hány cserét végeztünk a kirakás során.

2.20. Feladat

Osztálydiagram:

Egy XML dokumentumban különböző típusú tagok (node-ok) lehetnek, amelyek egymásba vannak ágyazva. Minden tagnak, és magának a dokumentumnak is lehetnek gyerekei, illetve szülője. Lehetőség van új gyerek hozzáadásra, egy gyerek törlésére, valamint az összes gyerek törlésére. A tagtípus lehet deklaráció, elem, illetve megjegyezés. A deklaráció kötelezően a dokumentum első eleme, amely tárolja a fájl kódolását és verzióját. Az elem rendelkezik névvel, valamint tetszőleges számú attribútummal. Az attribútumok kulcs-érték szövegpárokat tárolnak, így lehetőség van egy elem adott kulcsú attribútumának lekérdezésére, és felülírására. A megjegyzések csupán a megjegyzés szövegét tárolják. Ezen felül lehetőség van a dokumentum elmentésére, illetve betöltésére a fájlnév megadásával.

2.21. Feladat

Osztálydiagram:

Az egyetem karokból áll, azok pedig tanszékekből, mindegyik meghatározott költségvetéssel rendelkezik. Az egyetembe egyetemi polgárok járnak, akik lehetnek tanárok, hallgatók, illetve egyéb személyzet, akiket közvetlenül az egyetem alkalmaz, és lehetnek technikai, vagy adminisztrációs szerepkörben, de az egyetem nem különbözteti meg őket. A tanárok - akik lehetnek docensek, adjunktusok, illetve tanársegédek – a tanszék alkalmazásában állnak, meghatározott fizetéssel, illetve egyiket tanszékvezetőként alkalmazzák. Vannak gyakorlatvezetők is, akik lehetnek tanársegédek, illetve hallgatók is. A hallgatókat a tanárok tanítják, és ösztöndíjat kapnak az egyetemtől, ami a hallgatók tanulmányi átlagától függ.

2.22. Feladat

Osztálydiagram:

Az űrhajók legalább egy fedélzetből állnak, valamint egy fegyverrendszerből és egy meghajtórendszerből. A fedélzetek közül a legfontosabbak a híd, a raktár, a gyengélkedő és a gépház. Utóbbi a meghajtórendszert irányítja, amelynek maximális és aktuális sebessége van, továbbá tartozik hozzá legalább 2, legfeljebb 4 hajtómű, és a meghajtórendszer az, ami a fedélzeteket ellátja energiával. A fegyverrendszer egy vezérlőből és legalább egy fegyverből áll, ami lehet lézerágyú és torpedókilövő. A vezérlő - amelyet a hídról irányítanak – felel a torpedókilövők újratöltéséért. Az űrhajóknak páncélzata és pajzsa van, és amikor egy űrhajó megtámadja a másikat, egy fegyver csak akkor rongálja meg a másik űrhajót, ha a fegyver tűzereje nagyobb a pajzsnál.

2.23. Feladat

Osztálydiagram:

A BKV jegypénztárnál sorban várakoznak az utasok, akik meghatározott úti céllal rendelkeznek, illetve megvan a járatlista, amellyel a céljához eljuthatnak. A pénztárban két pénztáros dolgozik, akik kétféle bérletet - diák/nyugdíjas és felnőtt -, és háromféle jegyet - vonaljegy, szakaszjegy, átszállójegy - árusítanak. A bérletnek adott érvényességi ideje van, ha korábban váltják ki, akkor visszaváltható. A pénztárnak adott nyitva tartása van, a pénztáros pedig minden jegyből és bérletből egy meghatározott mennyiség. Ha valamelyik kifogy, a másiktól kell kérnie. Az utasok kérhetnek jegyet vagy bérletet, visszaválthatnak bérletet, illetve kérhetnek tanácsot is, hogy milyen járatokon közlekedjenek. Bérletet csak akkor kérhetnek, ha megfelelő igazolvánnyal rendelkeznek, ha pedig jegyet vesznek, azt az automatával érvényesíteniük kell. Ettől kezdve a

jegynek lesz egy érvényesítési ideje, illetve csak adott ideig használható csak fel, amely már az utazási eszköztől függ.

2.24. Feladat

Osztálydiagram:

Egy vidámparki hullámvasútról olyan adatokat tartanak nyilván, mint a sebesség és az izgalom. Egy hullámvasutat ki lehet nyitni, illetve be lehet zárni, és tartozik hozzá több szerelő. Mindegyiknek meghatározott időközönként kell ellenőriznie azt, és emiatt egyeztetnie kell a többivel, nehogy zavarják egymás munkáját. A hullámvasút sínekből, tartószerkezetből, egy megállóból, egy pénztárból és természetesen a szerelvényből áll. A sínek lehetnek egyvágányúak, kétvágányúak, illetve függőek, míg a tartószerkezet lehet fém, illetve fa. A szerelvény meghatározott számú kocsiból áll, és mindegyik kocsinak van egy ülésszáma – ennyien ülhetnek bele maximum, illetve egy hossza. A megálló hosszának akkorának kell lennie, hogy a minden kocsi elférjen rajta. Az utasoknál fontos tényezők a magasság és az émelygési szint. Az utasok a megállón keresztül sorban szállnak be a szerelvénybe – persze maximum annyian, ahányan befférnek a kocsikba, de legalább egy, különben nem indítanák el a vasutat -, de csak akkor vehetnek a pénztárban jegyet a hullámvasútra, ha elérik a 150cm-es magasságot, és ha valakinek út közben az émelygési szintje elér egy kritikus pontot, akkor ő kitaccsol a kocsiban.

2.25. Feladat

Osztálydiagram:

A naprendszereket egy csillag és valamennyi bolygó alkotja. A bolygók lehetnek lakottak, illetve lakatlanok, és a lakott bolygók körül keringhetnek űrállomások, egy körül legfeljebb 10, adott távolságra a felszíntől. Minden egyes űrállomásnak van valamekkora személyzete. Az űrállomások lehetnek katonaiak, civilek és kereskedelmiek. A katonai űrállomások valamennyi védelmi üteggel rendelkeznek, illetve 10-20 dokkal a hajók számára, és csak katonai űrhajókat fogadhatnak. Ezzel szemben a civil állomások csak 1-5 dokkal bírnak, és csak civil hajókat fogadhatnak. A kereskedelmiek pedig 5-10 dokkal rendelkeznek, illetve egy rakodóterülettel, és fogadhatnak katonai, valamint civil űrhajókat is. Az űrhajók kiszolgálása sorrendben történik. Az űrhajók tulajdonságai a személyzet száma, a sebesség és a páncélzat, a katonai űrhajóknál ezen felül a tűzerő.

2.26. Feladat

Osztálydiagram:

Egy város, amelynek meghatározott kiterjedése és lakossága van, lehet kis-, illetve nagyváros, továbbá a nagyvárosok egy külön típusa a világváros. Egy nagyvároshoz több kisebb város is tartozhat közigazgatásilag. A városokban közlekedési vállalatok működnek, egy városban legfeljebb kettő, míg egy vállalat több városban is jelen lehet. A vállalatok adott létszámú személyzettel és bizonyos költségvetéssel bírnak, amellyel új közlekedési eszközöket is vásárolhatnak, ha egy bizonyos határt túllépett a megtakarításuk. A közlekedési eszközök meghatározott útvonallal, egy vezetővel, utaskapacitással, illetve üzemeltetési költséggel rendelkeznek. A társaságok tulajdonában buszok, villamosok, metrók és repülők lehetnek. A buszok lehetnek helyiek, amelyek a városokban közlekednek, illetve távolságiak, amelyek a városok - akár több - között közlekednek. A villamosok és a metrók csak nagyvárosokban közlekednek, míg a repülők csak két világváros között repülnek. Persze a repüléshez szükség van repülőtérre, amely a város tulajdonában van, de csak akkor építenek, ha már a lakosság száma meghaladta a fél milliót. Ettől kezdve landolhatnak a gépek a repülőtereken.

2.27. Feladat

Osztálydiagram:

A vasúti szerelvények egy mozdonyból, és több, sorban egymás után következő kocsiból állnak. A mozdonyok lehetnek gőz, diesel, illetve villamos mozdonyok. Gőzmozdony esetén az első kocsinak egy szénszállítónak kell lennie. A kocsik lehetnek teherkocsik és személykocsik. Teherkocsik lehetnek konténerszállítók, rakteresek, illetve tartályosak. A személykocsik lehetnek fülkések és fülke nélküliek, illetve első és másodosztályúak. Van egy speciális kocsi, a bicikliszállító, ami egy másodosztályú személyszállító, és ugyanakkor egy rakteres teherkocsi. A kocsikat összecsatolják a többivel, illetve a mozdonnyal is, és a teherkocsikat megtöltik árutömbökkel, a kapacitásuknak megfelelően - azaz csak adott mennyiségig lehet beléjük árukat pakolni. Az árutömbök különböző típusúak lehetnek a benne tárolt áru típusának megfelelően, amelyekből adott mennyiség található bennük. A raktereseket különböző típusú áruval tölthetik fel, de egyet csak eggyel, és ez befolyásolja a raktér kialakítását.

2.28. Feladat

Osztálydiagram:

Egy fájlcserélő programban szerverekhez lehet csatlakozni, és a többi csatlakozott felhasználóval lehet fájlokat cserélni. A cserebere előtt egy fájlkezelő objektum eltárolja a megosztani kívánt fájlok listáját, miután feldolgozta azokat, hogy mások is el tudják érni őket. A fel és letöltését egy kimeneti és bemeneti csatorna felel. Előbbi egy kérést kap, majd lekérdezi a fájlkezelőt, hogy megtalálható-e az adott fájl a rendszerben, a kimeneti csatorna csak akkor indul el, ha a fájl bináris kódja megegyezik a kérés kódjával. A rendszer tartalmaz egy keresőt is, amellyel rákereshetünk fájlnevekre, méretre, illetve bináris kódra. A letöltés parancs kiadására létrejön a fájlhoz egy objektum, amely tartalmazza a 3 adatot, illetve a felhasználók listáját. Ezek az objektumok a letöltési listába kerülnek, amely folyamatosan figyeli, mely felhasználók vannak online a szervereken. Erre továbbá kiadható egy átnevezés, és egy "további források keresése" parancs. Utóbbi meghív egy új keresőt az adott bináris számmal. A bemeneti csatorna lekérdezi a letöltési listát a fájlokról, majd sorban meghívja a fájlokhoz tartozó listán található felhasználók kimeneti csatornáját, hogy lehet-e tőlük fájlt tölteni. Ha igen, akkor megindul a forgalom a két csatorna között.

3. Dinamikus ábrázolás

A dinamikus ábrázolás célja az általunk épített rendszerek vizsgálata a működés szempontjából. Sokszor a viselkedés, a dinamikus szerkezet ugyanolyan, vagy még nagyobb tervezést igényel, mint a statikus szerkezet. Természetesen a dinamikus vizsgálat előtt minden esetben ábrázolnunk kell statikusan, osztálydiagrammal a rendszert.

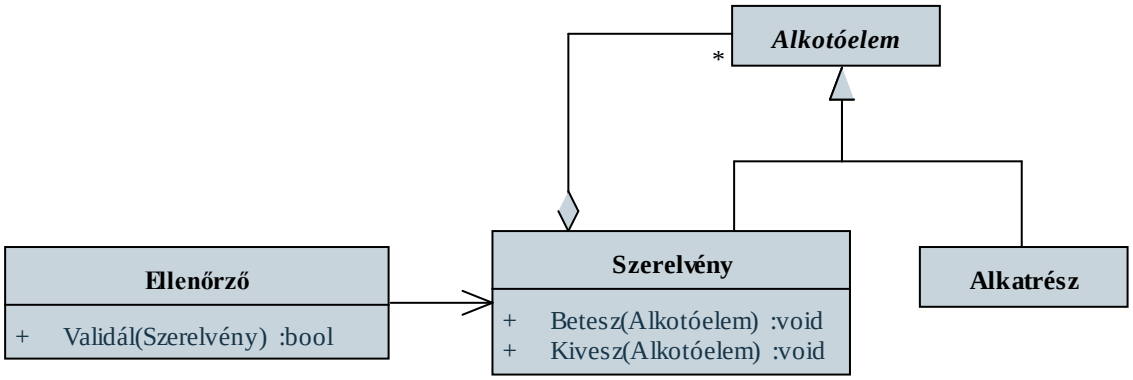
Az állapotdiagramban a rendszer kijelölt állapotai közötti kapcsolatokat keressük meg, illetve, hogy mivel feltételek, cselekmények és értékváltozások hatására keletkezett be az állapotváltozás.

Mintafeladat

Egy ipari környezetben egy rendszerrel szerelvényeket terveznek, amelyek különböző alkatrészekből, illetve további szerelvényekből (röviden alkotóelemekből) állnak össze. A szerelvényeket először megtervezik, majd gyártás alá kerülnek, végül kivonják őket a gyártásból. A kivont szerelvényeket azonban újra gyártás alá vonhatják, amennyiben a megmaradt készletek elfogynak. A tervezés egy összetett folyamat. Az újonnan létrehozott szerelvény konzisztensnek tekinthető, ám minden alkotóelem behelyezése, illetve kivétele után inkonzisztenssé válhat. Egy külön ellenőrző modul van a rendszerben, amely validálja a szerelvényt, így az ellenőrizetlen szerelvényből lehet konzisztens, vagy inkonzisztens szerelvény. Végezetül a konzisztens szerelvényt jóváhagyja egy mérnök, innentől tovább nem lehet módosítani, és gyártás alá kerülhet.

1. *Szerelvények statikus tervezése*

A *Szerelvény*, valamint az *Alkatrész* az *Alkotóelem* speciális esetei lesznek. Egy szerelvény tetszőlegesen sok alkotóelemet tárolhat, amelyeket a *Betesz*, illetve *Kivesz* műveletek segítségével módosíthatunk. A rendszerhez tartozik egy *Ellenőrző*, amely a *Validál* művelettel tudja ellenőrizni a szerelvény konzisztenciáját (11. ábra).

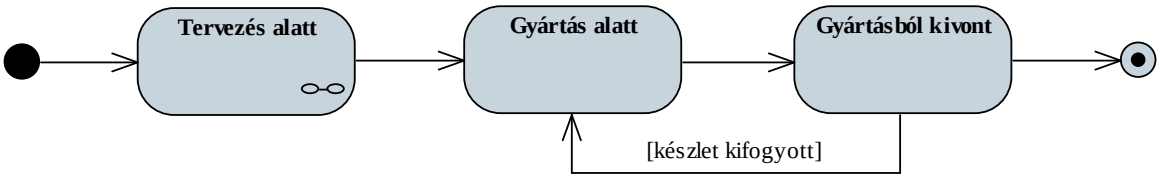


11. ábra: A szerelvények osztálydiagramja

2. *Szerelvények állapotai*

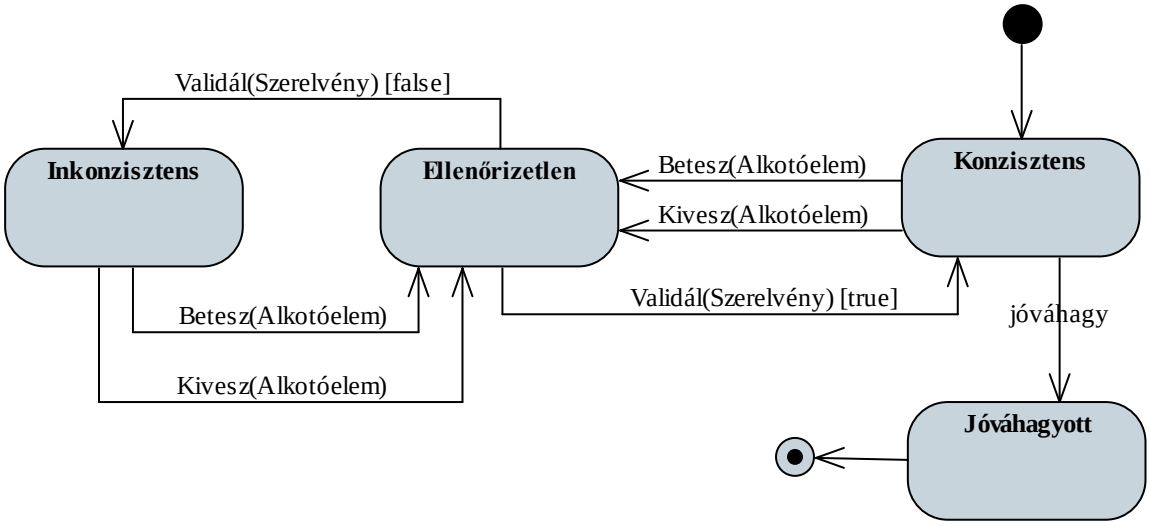
A szerelvény állapotait több szempontból is megközelíthetjük. A leghangsúlyosabb a gyártási folyamatban elfoglalt helye, hiszen ez mondja meg alapvetően milyen tevékenységi kör jellemző a szerelvényre. Ennek megfelelően három fő állapotot különböztethetünk meg, egy szerelvény lehet „tervezés alatt”, „gyártás alatt”,

valamint „gyártásból kivont”. Természetesen a kezdeti állapotból tervezés alá kerül a szerelvény, a végállapotba pedig a kivonásból kerül. Mivel a feladat nem ismertette az egyes állapotok közötti eseményeket, így csupán az átmeneteket jelöljük. Egyetlen ponton válthatunk vissza egy korábbi állapotba a folyamatban, amikor a kivont terméket újra gyártás alá helyezik. Ennek az eseménynek a feltétele, hogy a készlet kifogyjon, így ezt jelöljük a diagramon. A 12. ábrán látható a szerelvény teljes állapotdiagramja azzal a megjegyzéssel, hogy a tervezés egy összetett folyamat, így azt kompozit állapotként vesszük fel, és kibontjuk.



12. ábra: A szerelvény állapotdiagramja

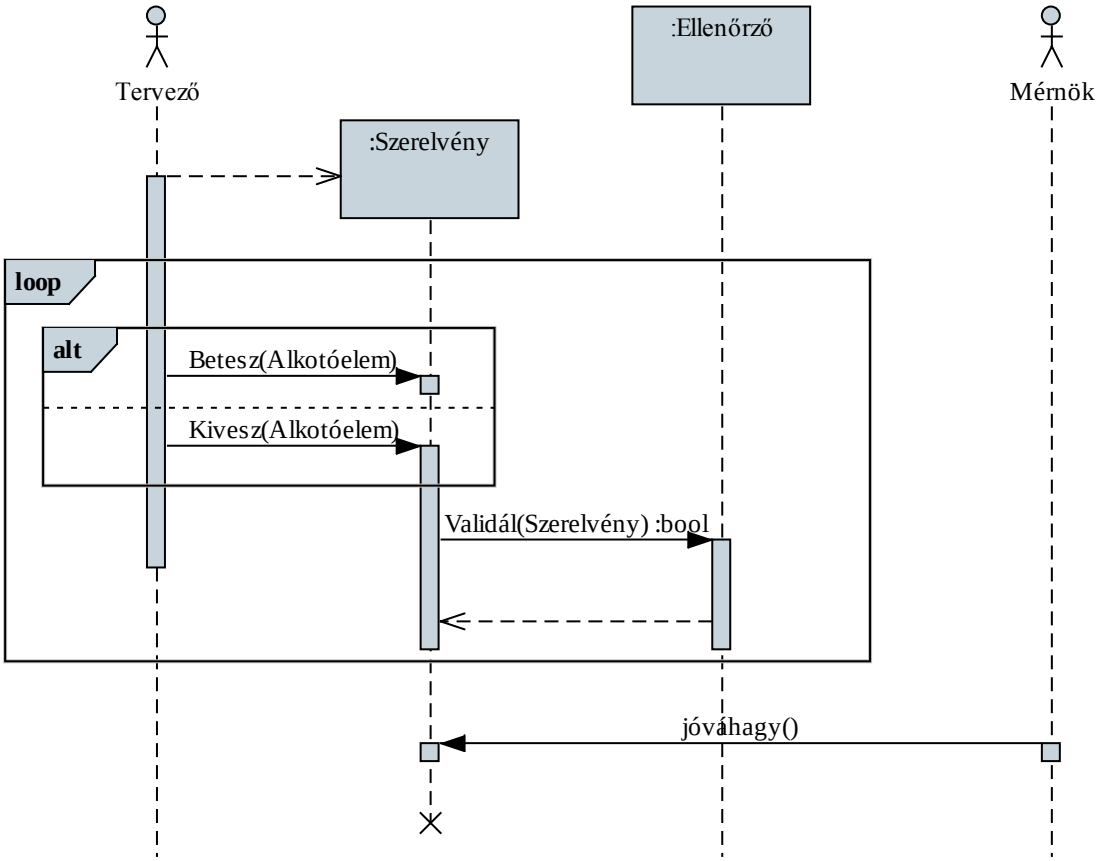
A tervezés állapotát a konzisztencia és a jóváhagyottság szempontjából közelítjük meg. Bevezetjük a „konzisztens”, illetve „nem konzisztens” állapotokat, amelyek között alkotóelemek behelyezésével/kivételével, illetve a validáció végrehajtásával tudunk váltani. Mivel ez egy több lépéses folyamat, és több kimenete is lehetséges, érdemes közbeiktatni egy további, „ellenőrizetlen” állapotot, amely a módosítást követően, de a validáció végrehajtása előtt áll fenn, így bármely állapotban is volt a szerelvény, a módosító műveletek az ellenőrizetlen állatba viszik át, amelyben természetesen nem módosíthatunk. Innen validáció segítségével léphetünk át a konzisztens, vagy inkonzisztens állapotokba a validáció eredménye függvényében. Ezt, mint feltételt vehetjük fel az eseményhez. A konzisztens állapotból jóváhagyással léphetünk a „jóváhagyott” állapotba, ahol már semmilyen validáció, vagy módosítás nem érheti a szerelvényt. Végezetül, a szerelvény kezdeti állapota a konzisztens állapotba vehet, míg a végállapotot (amely egyúttal a „tervezés alatt” állapotból való kilépés) a jóváhagyott állapotból érhetjük el (13. ábra).



13. ábra: A „tervezés alatt” állapot kifejtése

3. *A tervezés szekvenciája*

A tervezés folyamata jól ábrázolható szekvenciaként is. A szekvenciában két aktor vesz részt, a *Tervező*, illetve a *Mérnök*, valamint a *Szerelvény*, illetve az *Ellenőrző* egy-egy példánya. A tervező elsőként létrehozza a szerelvényt, így ez az életvonal később indul (a további résztvevők már kezdetben léteztek). A tervezés jó része egy ismétlődő eseménysorozat, amelyet ciklus (loop) segítségével ábrázolhatunk. Ezen belül először módosítjuk a szerelvényt, amely lehet alkotóelem behelyezés, illetve kivétel. Ezeket természetesen a tervező végzi, és egy elágazás (alt) segítségével ábrázolhatóak. A módosítást követi az ellenőrzés. Mivel ez automatikus, kezdeményezheti a szerelvény. Itt külön felhívhatjuk a figyelmet, hogy a visszatérési értéket figyelembe kell vennünk. A ciklusból kilépve a jóváhagyás következik, és ezzel végeztünk a tervezéssel.



13. ábra: A tervezés szekvenciadiagramja

3.1. Feladat

Osztálydiagram és állapotdiagram:

Egy videólejátszó program egy vezérlőből és egy megjelenítőből áll, amelyeket 4 nyomógomb segítségével irányítunk. Ennek megfelelően lehet lejátszani egy filmet, előrepörget, visszapörget, valamint megállítani. A pörgetés csak lejátszás közben működik, leállítani bármikor le lehet a videót. A megjelenítő minden esetben létrehozza a képet (működik), kivéve, ha leállítjuk a lejátszást.

3.2. Feladat

Osztálydiagram és állapotdiagram:

Egy repülő vagy a földön várakozik, vagy a levegőben repül. Csak akkor szállhat fel egy repülőtérről, vagy szállhat le egy adott repülőtérre, ha a torony engedélyezi a felszállást, és a torony repülés közben is folyamatosan kommunikál a repülővel.

3.3. Feladat

Osztálydiagram és állapotdiagram:

Egy útkereszteződésnél két szemaforral szabályozzák a forgalmat. A szemafor zöld, vagy piros lehet, válthat a kettő között, továbbá meg van adva, hogy mennyi ideig kell az állapotot tartania. A szemafor csak akkor vált, ha az idő lejárt. A szemafort lehet manuálisan is váltani, ekkor az idő újraindul. Kezdetben az egyik szemafor zöld, a másik piros.

3.4. Feladat

Osztálydiagram és állapotdiagram:

Egy egyetemi hallgató rendelkezik valamennyi pénzmennyiséggel, tanulhat, pihenhet, szórakozhat, illetve kaphat ösztöndíjat, továbbá lehet friss, fáradt, illetve másnapos. Ha a hallgató tanulás után még mindig friss, és van pénze, akkor elmegy szórakozni, különben pihen. Pihenés után, ha van pénze, akkor elmehet szórakozni, ha nincs, akkor tanul. A tanulásnak két formája van, az otthoni tanulás, illetve az óralátogatás. Utóbbit csak akkor végezheti, ha nem másnapos. A hallgató addig szórakozik, amíg van pénze, és azután mindenképpen pihenni megy.

3.5. Feladat

Osztálydiagram és állapotdiagram:

Egy számítógép egy processzorból, egy memóriából és egy háttértárból áll, és programok futnak rajta. A processzor és a háttértár lehet foglalt, vagy szabad, a memória pedig szabad, olvasás, illetve írás alatti. Mindegyiket egyszerre csak egy program használhatja. Egy program az elindítás után a háttértárról töltődik be, majd a memóriába kerül, ekkor már futó állapotban van. Futás alatt előbb olvas a memóriából, majd a számításokat végez (használja a processzort), végül visszaír a memóriába, majd újra olvas. Futásból kimentő állapotba csak írás után kerülhet, ekkor visszaír a háttértárra, végül terminál.

3.6. Feladat

Osztálydiagram és állapotdiagram:

Az étkező filozófusok egy kerek asztal mellett ülnek. Minden filozófusnak van egy tányérja, valamint egy villája. A filozofálhat, illetve ehet, azonban ehhez két evőeszközre is szüksége van, így amennyiben egy filozófus étkezik, a mellette ülő filozófusok nem étkezhetnek. A filozófusok lehetnek mohók, vagy türelmesek. Mindegyiknek van egy éhségszintje, ami ha egy adott határt elér, akkor megpróbálnak enni, de ha nem sikerül mindkét villát felvenniük, csak az egyiket, akkor két lehetőség van. Ha mohók, akkor nem engedik el a villát, hanem megvárják, hogy a szomszéd elengedje az övét. Ha türelmesek, akkor visszarakják azt a villát, amit sikerült felvenniük, és egy ideig újra filozofálnak. Ha az éhség eléri a kritikus szintet, akkor a filozófus meghal.

3.7. Feladat

Osztálydiagram és állapotdiagram:

Egy banki adatbázisrendszerben négyféle szerver található, egy központi, egy tartalék, több helyi, illetve két biztonsági. A helyi szervereken adatbázis-kezelők futnak, amelyeken található adatbázisokat a biztonsági szerverek adott időközönként saját tárhelyükre elmentik. Ekkor az

adatbázisokon futó összes további művelet megszakad, majd a mentés után újra hozzáférhetővé válik, és minden művelet onnan folytatódik, ahol abbamaradt. A rendszert a központi gép vezérli, és ha az meghibásodik, a helyére lép azonnal a tartalék szerver. Amint a központi szervert megjavítják, az visszalép működésbe, a tartalék szerver pedig figyelő üzemmódba kerül. Az adatbázisokhoz kliens gépek férnek hozzá, ennek folytán lehetnek írás, illetve olvasás alatt, vagy használaton kívül. Természetesen írni egy adott adatbázist csak akkor lehet, ha semelyik más kapcsolódó kliens nem írja, vagy olvassa azt.

3.8. Feladat

Osztálydiagram és állapotdiagram:

A metróvonalon legalább két állomás található, minden állomáson két lámpa van. A vonalon szerelvények járnak. Az állomás két lámpája közül az egyiket a szerelvények beengedésének, a másikat a szerelvények kilépésének (továbbhaladásának) a vezérlésére használjuk. A szerelvények sorban haladnak a vonal állomásain az alábbiak szerint. Egy állomáson legfeljebb csak egy szerelvény tartózkodhat. Ha az állomáson szerelvény tartózkodik, akkor a beengedő lámpa piros. Szerelvény csak akkor mehet az állomásra, ha a beengedő lámpa zöld. A szerelvény egy állomáson adott ideig tartózkodik, és ha ez az idő letelt és a kivezető lámpa zöld, akkor elhagyja az állomást. Ekkor a kivezető lámpa pirosra vált, és adott ideig piros marad. Ha az idő letelt, zöldre vált.

3.9. Feladat

Osztálydiagram és szekvenciadiagram:

Egy űrhajó megérkezik az űrállomáshoz. Leszállási engedélyt kér, majd miután megkapja, landol. Később elhagyja az állomást.

3.10. Feladat

Osztálydiagram és szekvenciadiagram:

Egy benzinkúti tankolásnál jön egy autó, amely tankol, majd fizet a pénztárnál, és távozik. A fizetés közben érkezik egy másik autó, amely tankol, letörli a szélvédőjét, majd ezután fizet a pénztárnál, és szintén távozik.

3.11. Feladat

Osztálydiagram és szekvenciadiagram:

Egy telefonhívás keretében a hívó felemeli a kagylót, majd a szabad jelzésre tárcsázza a kívánt számot. Ezután a telefon kapcsolja a hívottat, aki bejelentkezik, és így megkezdődik a beszélgetés.

3.12. Feladat

Osztálydiagram és szekvenciadiagram:

Egy programozó elkészít egy programot, majd bemutatja azt a vevőnek. A vevő ekkor esetleg kérhet további funkciókat a programtól, amikkel a programozónak ki kell egészítenie a programot. Ha ez megtörtént, a vevő kifizeti a programozónak a program árát.

3.13. Feladat

Osztálydiagram és szekvenciadiagram:

Egy vásárlás során a vásárló bemegy az üzletbe, vesz magának egy kosarat, vagy egy bevásárlókocsit, attól függően, milyen hosszú a bevásárlólistája. A vásárló addig pakolgatja az árukat kocsijába, amíg meg nem talál mindent a listáján, majd a kasszához megy, ahol várakozik, majd fizet. Miután megkapta a visszajárót, elhagyja az üzletet.

3.14. Feladat

Osztálydiagram, állapotdiagram és szekvenciadiagram:

Az autóriasztó rendszer 3-5 ajtóból, egy riasztó egységből és egy érzékelőből áll. Az ajtók nyithatók és zárhatók, az érzékelő az autóban zajló mozgásokat érzékeli és továbbítja a riasztónak. A riasztót egy irányítóval lehet be- illetve kikapcsolni. Ha a riasztó be van kapcsolva és valamelyik ajtót kinyitják vagy az érzékelő mozgást jelez, akkor a riasztó riaszt. Ezt kikapcsolással lehet megszüntetni. A riasztót csak akkor lehet bekapcsolni, ha minden ajtó zárva van.

3.15. Feladat

Osztálydiagram, állapotdiagram és szekvenciadiagram:

Záróvizsgát egy bizottságnál tehetnek a diákok. A bizottság 3 tagból áll, a diákok száma tetszőleges. A záróvizsgára egy teremben kerül sor, ahol egyszerre legfeljebb adott számú diák tartózkodhat. A terem kezdetben üres. Diák akkor léphet be a terembe, ha van még hely és a bizottságból legalább egy tag bent van. Belépés előtt a diákok a terem előtt várakoznak. A tagok bármikor bemehetnek a terembe, de elhagyni csak úgy, hogy ha van bent diák legalább egy tagnak bent kell lennie és felelet esetén legalább kettőnek. Belépés után a diák felkészül, majd ha a bizottság szabad (nem felel senki) és két tagja jelen van, elkezd felelni. A felelet végeztével a diák elhagyja a termet.

3.16. Feladat

Osztálydiagram, állapotdiagram és szekvenciadiagram:

Egy mozi mozitermekből, egy pénztárból és egy büféből áll. A moziban filmeket vetítenek, és a moziba nézők járnak. A pénztárnak és a büfének is adott kapacitása van, vagyis, hogy hány embert tudnak egyszerre kiszolgálni. Egy terem lehet zárva, nyitva, illetve foglalt, ha éppen filmet vetítenek benne. Kezdetben zárva van, a vetítés előtt 15 perccel megnyitják, majd a vetítés után bezárják, ha már senki sem tartózkodik a teremben. A néző először a pénztárhoz áll be, és várakoznia kell, amíg a pénztárban nem lesz hely, és ő az első a várakozók közül, ekkor megvásárolja a jegyét - amely adott sorszámú teremhez tartozik -, majd beáll a büféhez, ahol szintén várakoznia kell, ha vannak előtte. Miután a büfében is vásárolt, várakoznia kell, amíg meg nem nyitják azt a termet, ahova a jegye szól. Ha megnyitják, bemegy, végignézi a filmet, majd távozik a moziból.

3.17. Feladat

Osztálydiagram és állapotdiagram:

Egy számítógépes gépteremhez egymás után érkeznek csoportok, amelyek használni szeretnék azt. A géptermet egyszerre csak egy csoport használhatja, a többi csoport kint várakozik. Azok a csoportok, amelyek zárthelyi írására szeretnék használni a géptermet előnyt élveznek a többi csoporttal szemben. Azonos prioritású csoportok közül a korábban érkezettet kell beengedni.

Szekvenciadiagram:

Az első csoport érkezik először, és amíg az a gépteremben dolgozik, megérkezik a második, majd a harmadik csoport. A harmadik csoport zárthelyit szeretne írni a gépteremben.

3.18. Feladat

Osztálydiagram:

Az orvosi rendelő egy váróból, két öltözőből, egy vizsgálóból, valamint az irodából áll. Az orvosi rendelőbe egy orvos és egy nővér dolgozik. A rendelőbe betegek járnak, akik sorszámmal rendelkeznek, illetve vannak az időpontra behívott betegek, akik elsőbbséget élveznek.

Állapotdiagram:
A betegek először sorszámot húznak, majd leülnek a váróba, amíg nem lesz szabad öltöző. A szabad öltözőt mindig a legalacsonyabb sorszámú beteg foglalja el, persze ha vannak időpontos betegek, akkor közülük megy be a legalacsonyabb sorszámú az éppen szabad öltözőbe - amely ekkor foglalttá válik. Az öltözőben levetkőzik, majd a nővér szól neki, ekkor bemegy az irodába az adatait bediktálni, és várja, amíg az orvos nem végzett az előző beteggel. Ezután a vizsgálóban megvizsgálják, majd visszajön az irodába, ahol megvárja a receptjét. Ha megkapta, és kifizette az orvost, visszamegy az öltözőbe felöltözni – miután felöltözött, az öltöző újra szabaddá válik -, majd elhagyja a rendelőt.
A nővér tehát szól az öltözőbe - ha vannak benne -, felveszi a beteg adatait, majd amíg a beteg a vizsgálóban van, játszik a számítógépen, amíg az orvos nem végez. Ekkor felírja a receptet, elintézi a fizetséget, és szól a következő betegnek a másik öltözőben.
Az orvos a vizsgálóban vár, és ha új beteg jön az irodába, előbb meghallgatja a beteg panaszait, majd kivizsgálja, végül, lediktálja a nővérnek a recepteket, majd visszatér a vizsgálóba.

Szekvenciadiagram:
A beteg megérkezik a váróba, húz egy számot, majd ha van előtte beteg leül. Nemsokára bemegy az első öltözőbe, levetkőzik, vár egy keveset, míg a nővér beszólítja, aki felveszi az adatait. Ezután befárad a vizsgálóba, ahol az orvos megvizsgálja, majd ha indokoltnak látja, felíratja a nővérrel a receptet. A beteg fizet, felöltözik, majd távozik.

3.19. Feladat

Osztálydiagram:
A metróvonalon metrószerelvények közlekednek, amelyek metrókocsikból állnak. A metrókocsik lehetnek vezetőfülkések, illetve szabványosak. A metrókocsikat összekapcsolják, pontosan annyit, hogy a szerelvény hossza megegyezzen az állomás hosszával. Az állomásra egyszerre két szerelvény érkezhet – ellentétes irányból -, a szerelvények érkezési idejét egy óra jelzi, az állomáson tartózkodhat egy jegyellenőr is. Az utasok meghatározott célállomásra szeretnének eljutni, és közlekedhetnek jeggyel, bérlettel, illetve bliccelhetnek. Az állomáshoz tartozik egy jegyellenőr, így a bliccelő utasnak előbb egy elterelő hadműveletet kell alkalmaznia, hogy közlekedhessen a metrón, vagy várnia kell, amíg az ellenőr elmegy a mellékhelységre.

Állapotdiagram:
Az utasok megérkeznek az állomásra, érvényesítik a jegyüket, ha van nekik, majd lemennek a lefelé haladó lépcsővel, és várnak a megfelelő irányba tartó szerelvényre, addig, amíg egy olyan nem jön, ahol a helyek legalább 10%-a szabad. Ezután beszállnak a szerelvényre, majd a célállomásnál kiszállnak. Ha bliccelnek, és van jegyellenőr, akkor alkalmazzák az elterelő műveletet, vagy várakoznak, és csak akkor mennek a szerelvényhez, ha az ellenőr már nincs ott. A jegyellenőr tehát vagy figyeli az utasokat, vagy elterelték a figyelmét, vagy a mellékhelyiségben tartózkodik.

Szekvenciadiagram:
Az első utas megérkezik, megmutatja az ellenőrnek a jegyét, megvárja a metrószerelvényt, majd felszáll az egyik kocsiba. Addig utazik a metróval, amíg a célállomásához nem ért, ekkor leszáll. A második utas bliccel, ezért eltereli a jegyellenőr figyelmét, majd szintén felszáll az előbbi szerelvényre, és leszáll az ő célállomásánál.

3.20. Feladat

Osztálydiagram:

Egy szigorlaton a hallgatókat egy bizottság vizsgáztatja egy tanteremben. A szigorlatnak időpontja van, valamint egy maximális létszáma. A bizottság 3 tagból áll, akik lehetnek docensek, adjunktusok, illetve tanársegédek, egyikük a bizottság vezetője. A szigorlat egy tanteremben zajlik, amely adott maximális hallgatószámot tud befogadni, ezért maximum annyian tartózkodhatnak a teremben.

Állapotdiagram:

A terem lehet üres, majd miután bemegy legalább egy bizottsági tag, használatba kerül, illetve lehet tele, ha már minden hely a teremben foglalt. Miután az összes bizottsági tag elhagyta a termet, újra üres lesz. A hallgatók egymás után mehetnek be a terembe, ha az használatban van, de nincs tele. Ezt követően a hallgató tételt húz, készül, majd ha a bizottság szabad, és legalább két tagja a teremben van, akkor vizsgázhat. A vizsgát követően elhagyja a termet. A bizottsági tagok bejönnek a terembe, de később el is hagyhatják azt, majd újra visszajöhetnek, amikor a teremben vannak, akkor várakoznak, vagy vizsgáztatnak. Egy tag csak akkor mehet ki a teremből, ha a bizottság nem vizsgáztat senkit, vagy rajta kívül ketten maradnak befejezni a vizsgáztatást. Amíg van hallgató a teremben, addig legalább egy bizottsági tagnak mindig a teremben kell lennie.

Szekvenciadiagram:

A bizottsági tagok várakoznak, majd egy hallgató belép a terembe, tételt húz, és elkezd készülődni. Ha úgy gondolja, nem tudja elmondani a témáját, akkor húzhat másik tételt. A készülés után a hallgató jelez a bizottságnak, hogy elkészült. A bizottság szólítja őt, majd addig kérdezi, amíg el nem tudja dönteni, milyen jegyet kapjon. A vizsga végén a bizottság jegyet ad a hallgatónak, aki ezután távozik a teremből.

3.21. Feladat

Osztálydiagram:

Egy szövegszerkesztő programban van egy szövegszerkesztő képernyő, valamint egy menüsor és egy állapotsor. A szövegszerkesztőknek két fajtája létezik, az egyablakos, illetve a többablakos. Előbbinél csak egy dokumentumot lehet egyszerre megnyitni, míg utóbbinál többet is, és ekkor persze mindegyikhez külön szerkesztőképernyő kell. A menüben olyan funkciók találhatók, mint a megnyitás, mentés, kilépés, illetve persze vannak a dokumentumkezelő funkciók, amelyeket minden megnyitott dokumentumnál külön objektum kezel. Egyike ezek funkcióknak a visszavonási művelet. Ez egy, a dokumentumkezelőhöz tartozó láncolt listában tárolja a korábbi műveleteket, ahol a listaelemek tartalmazzák a listaművelet nevét, a változtatás helyét a szövegben, illetve az általa megváltoztatott szövegrész korábbi változatát. Ezenkívül a további dokumentumszerkesztő műveleteket az átszínezés, betűtípusváltás, a tabulálás, a hasábok alakítása, az oldalra zárás, illetve a másolás, kivágás és beszúrás. A szövegszerkesztő persze több dokumentumtípust is tud kezelni, és ennek megfelelően a dokumentumkezelő ehhez tud alkalmazkodni.

Állapotdiagram:

A szövegszerkesztővel a dokumentumokat először megnyitjuk, majd azoknak létrejön a típusnak megfelelő kezelő objektuma, ettől kezdve szerkeszthető lesz a dokumentum. Szerkesztés után elmenthetjük a dokumentumot, de ez nem kötelező. A szerkesztő műveletek először kiadás alatt vannak, majd a program elvégzi őket. Ennek folyományaként a szerkesztés szakasza alatt a művelet kiadása után a dokumentum nem szerkeszthető állapotba kerül, és csak vált vissza, ha a

művelete sikeresen befejeződött. Ha nem sikerült elvégezni a műveletet, a program újra próbálkozik, de eztán már mindenképpen újra szerkeszthetővé válik a dokumentum, akár sikerül a művelet, akár nem. A szövegszerkesztő állapota ezért nyilván a megnyitott dokumentumok függvénye.

Szekvenciadiagram:
A szövegszerkesztő programunkkal megnyitunk egy dokumentumot, ekkor létrejön a neki megfelelő kezelő objektum. Ezután az objektum segítségével végrehajtunk rajta egy beillesztés, két tabulálás, és egy másolás műveltet. Ezután megnyitunk még egy dokumentumot, amelynek kezelője segítségével beszúrjuk az előbb kimásolt részt. Ezután mindkét dokumentumot elmentjük, majd bezárjuk.