Szoftverfejlesztés

Az UML modellezés alapjai

Készítette: Angster Erzsébet, 2003. június

Munkaanyag, 0.5 verzió

Tartalomjegyzék

Ebben a részben a példákat lényegében a következő esettanulmányokból merítjük:

KissDraw – Rajzoló program

A KissDraw egy rajzolóprogram, mely segítségével különböző színű síkidomok (egyenes, téglalap, ellipszis) és szabadkézi rajzok tehetők egy "rajzlapra". A rajzlap háttérszíne megadható. Az egyes síkidomok kijelölhetők és törölhetők. A rajzlapokat lemezre lehet menteni.

Mikro – Egyperces mikrohullámú sütő

Egy egyszerű mikrohullámú sütőt kell modellezni. A sütőnek csak légkeveréses főzési üzemmódja van. A vezérlőpanelen három gomb található: a Perc plusz, Töröl és Ajtó. Főzés csak csukott ajtó mellett történhet. Ha nyitva van az ajtó, illetve a főzés ideje alatt a sütőben ég egy kis lámpa.

Szerviz

Szervizünk számos ügyféllel áll kapcsolatban. Az ügyfeleink a szerviz telephelyére behozzák a javításra vagy karbantartásra váró gépkocsikat. Az ügyintézőnk rögzíti az ügyfél, valamint a gépkocsi adatait, feljegyzi az elvégzendő tevékenységeket (ez egy vagy több szabad szöveges leírás), valamint a vállalási határidőt – a határidőt az ügyintéző szakmai tapasztalata alapján becsli meg.

A szerviz belső működését nem követi a program, kivéve a beépített alkatrészek illetve az elvégzett tevékenységek és a hozzájuk tartozó rezsi órák regisztrálását. A javítás elkészülte után az ügyintéző átadja az autót a megrendelőnek, s számlát nyomtat a program. A számlán szerepelnek a beépített alkatrészek, illetve az összes ráfordított munkaidő az eladási áron kiszámlázva.

Az alkalmazást fel kell arra készíteni, hogy akár az ügyfél neve, akár a gépkocsi rendszáma alapján vissza tudjon keresni. Egy ügyfélhez akár több gépkocsi is tartozhat. Az ügyfél személyes adatain (név, cím, telefon, mail) túl kezelni kell a számlázási névre, címre vonatkozó adatokat is.

A szerviz-dolgozó számára az alkalmazás ki tudja listázni az elvállat javításokat. Megadhatja, hogy az egyes autókhoz milyen alkatrészt használt fel, illetve ténylegesen mennyit dolgozott a javítással. Lekérdezheti az illető autó történetét: mikor, milyen javításokat végeztek, milyen alkatrészeket építettek be.

Az alkalmazás legyen képes papír alapú üdvözlő levelet, illetve mailt is küldeni az ügyfeleknek. A program képes bizonyos vezetői információk előállítására is.

TanLev – Tanári levelező program

Egy olyan szoftvert kell készíteni, mely

- információt ad a Gábor Dénes Főiskola (GDF) kihelyezett központjairól, a főiskola tantárgyairól, azok tantárgyfelelőseiről és a tantárgyakat tanító tanárokról;
- a tantárgyfelelősök leveleket (információt, értesítést, figyelmeztetést) küldhetnek a tanároknak és a központoknak, mely leveleket megtekintés céljából vissza is lehet keresni.

Az esettanulmányok fejlesztői és felhasználói dokumentációi a mellékletben megtalálhatók (majd).

Ingatlanközvetítő

A szoftver segítségével lakossági ingatlanok (telkek, családi házak, lakások) eladását, illetve bérbeadását szeretnénk közvetíteni. Ipari ingatlanokkal nem foglalkozunk. A rendszer tárolja a hirdető személy (tulajdonos vagy ügynök) által kínált ingatlan adatait. Az egyes ingatlanról azokat az adatokat szeretnénk jegyezni, amelyekre az ingatlant kereső kíváncsi lehet. Az egyes ingatlanokról képek is tárolhatók. Az ingatlant a rendszerben módosítani is lehet, illetve ki is lehet törölni.

A szoftver legyen alkalmas szerződések megkötésére és tárolására is. A rendszernek gondoskodnia kell az adatok időszakos archíválásáról, illetve az archív adatok visszakereséséről. A kereső személyeket a rendszer nem tárolja.

Az ingatlanokból a kereső saját szempontjai szerint válogathat, távoli eléréssel, webes feületen. A kiválasztott ingatlanok bizonyos adatairól nyomtatott listát lehet készíteni. A kiválasztott ingatlanok megtekintése végett be kell fáradni az irodába.

1. Az UML koncepcionális modellje

Ebben a fejezetben röviden ismertetjük a modell és nézet fogalmát általában, és az UML-ben. Ezután rendszerezzük az UML építőköveit: a modellelemeket, az azok közötti kapcsolatokat és a diagramokat. A fejezet célja, hogy elősegítse a modellezésben és az UML diagramokban való eligazodás készségét, és egy alapot adjon a könyv további fejezetinek használatához. A fejezet olyan értelemben referencia jellegű, hogy felsorolja a könyv által használt UML diagramokat. E felsorolás azonban koránt sem teljes – az UML teljes leírása az *OMG Unified Modeling Language Specification* dokumentumban található.

A diagramokról és az azokat jellemző modellelemekről külön fejezetek szólnak majd. A modellekkel és nézetekkel *A fejlesztés módszertana* rész foglalkozik részletesen.

1.1. Modell, modellelem, nézet, diagram

Egy bonyolult rendszert teljes egészében és részletességében lehetetlen egyszerre látni, érteni, illetve leírni. A rendszer megértéséhez és dokumentálásához a rendszert nézetekre és részekre (csoportokra) szét kell szedni, és az így elkülönített dolgokat különböző absztrakciós szinteken külön-külön kell modellezni, leírni. Részletkérdés, hogy a bontást mivel kezdjük: a nézeteken és csoportokon belül adjuk meg az absztrakciós szinteket; vagy az absztrakciós szinteken belül adjuk meg a különböző nézeteket és csoportokat.

A modell és nézet fogalmának tisztázásához egy klasszikus példát fogunk használni, a házat. A ház kinézetének és működésének megértéséhez a házat modellezni szokták.

A teljes rendszer a ház, ennek modelljei például:

- Engedélyezési tervdokumentáció. Modellelemek: fal, ajtó, tető stb.
- Kivitelezési tervdokumentáció. Modellelemek: fal, ajtó, tető stb, de itt sokkal részletesebben; valamint a részletezéshez szükséges további elemek, mint gázcső, villanyvezeték, központi porszívó stb.

A fenti modellek (itt speciális tervek) túl bonyolultak ahhoz, hogy azokon minden információ egyszerre szerepelhessen. A terveket különböző nézetekben kell megvilágítani. Egy háztervnek számos nézete lehetséges: látványterv, alaprajzok, statikai terv, villamos terv stb.

A modellek diagramokból és leírásokból állnak. A két házterv diagramjai: Déli homlokzat, Pince alaprajza, Földszinti alaprajz, stb., leírásai az építészeti leírás, műszaki leírás, statisztikai igazoló számítások, tervezői nyilatkozat stb.

Modell: A modell egy rendszer (bonyolult probléma vagy szerkezet) absztrakciója, amely a megértést és a kezelhetőséget célozza. A modell az adott rendszert egy bizonyos absztrakciós szinten teljes mértékben leírja.

Nézet: A rendszernek lehetnek nézetei. Az egyes nézetek a rendszert más-más szemszögből világítják meg, más-más oldalról mutatják.

A modell és a nézet a rendszer egymástól független megközelítései.

Modellelem: A modell alkotóelemei a modellelemek. Minden egyes modellelemnek más a célja, jelölése, és más-más szabályok vonatkoznak rá.

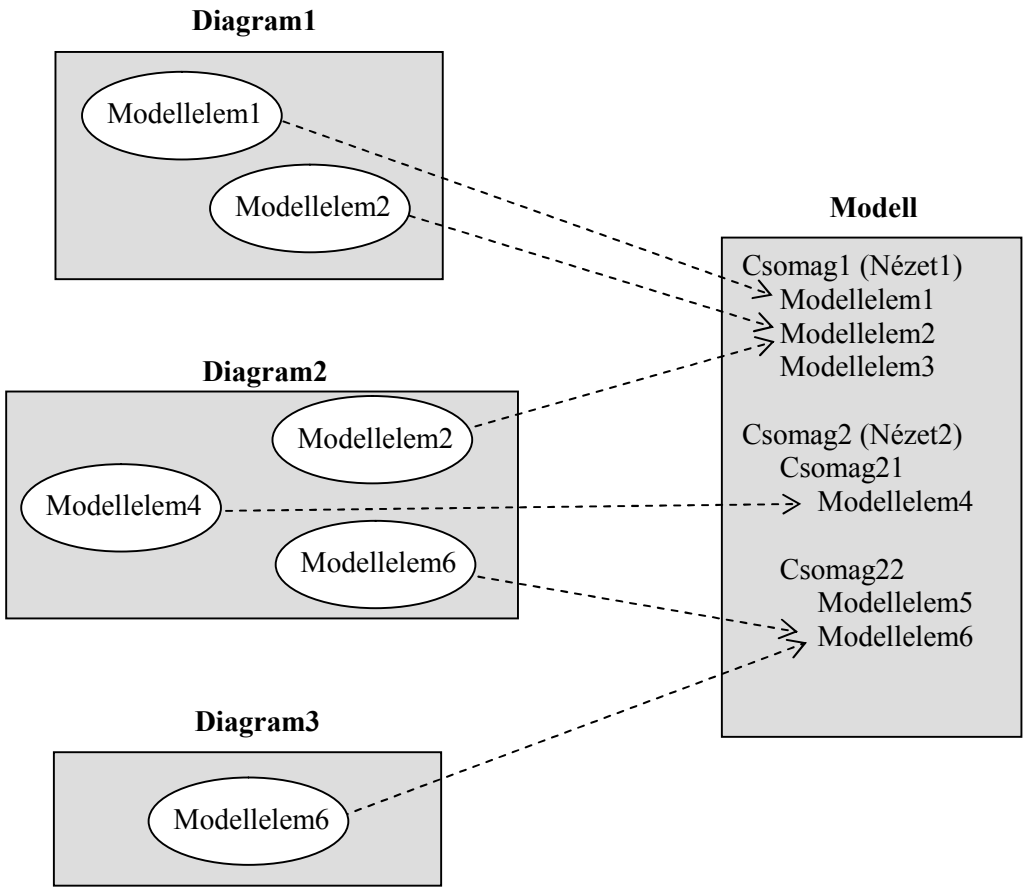
A modellelem tulajdonságai:

- A modellelemek csoportosíthatók. A modellelemek csoportjait csomagnak (package) nevezzük. A nézet a modell legfelső szintű csoportja.
- A modellelem a modellben egyértelműen azonosítható. A modellelemek csomagneveikkel minősíthetők.
- A modellelemnek lehet szereotípusa (fajtája) és lehetnek tulajdonságai.
- A modellelemek között kapcsolatok definiálhatók.

Diagram: A diagramokat modellelemek és az azok közötti kapcsolatok alkotják. Egy diagramra bármely modellelem rátehető, és egy modellelem több diagramon is szerepelhet. Ha egy modellelemet megszüntetünk, akkor az az összes diagramról eltűnik. Két modellelem közötti kapcsolat nem feltétlenül látszik egy diagramon.

Egy modell tartalmazhat diagramokat, leírásokat, és bármit, ami a megértést elősegíti.

Megjegyzés: A Magyar Értelmező Szótár szerint a tudományban „a modell valamely jelenség, rendszer jellemzőit, összefüggéseit kifejező, ábrázoló, jelképező logikai formula.”



Modell, modellelem, nézet, diagram

A modell nézetei, avagy a nézet modelljei?

A rendszer egy modelljének lehetnek különböző nézetei. A ház esetében az Engedélyezési „modell” például tartalmaz egy nagyvonalú statikai „nézetet” – ez csak az ún. architekturálisan fontos számításokat tartalmazza. De a Kivitelezési „modell” is tartalmaz statikai „nézetet”, ami már egészen pontos számításokat tartalmaz. Ha valaki csak a ház statikájára kíváncsi, akkor nyugodtan be lehet sorolni a statikai „nézet” alá a két modellt, azok megfelelő részeit.

A mellékelt ábrán az első megoldást választottuk. Az ábra jobb oldalán egyetlen modell látható. A modellnek 6 eleme van, csomagokba osztva. A legfelső szinten 2 csomag van, a második csomagnak két alcsomagja van. A diagramokra a modell elemeiből válogattunk: a Modellelem2 két diagramon is szerepel (Diagram1 és Diagram2), a Modellelem5 nem szerepel egyetlen diagramon sem. A szaggatott nyíl függőséget jelent: A diagram megfelelő eleme a modellben nyilvántartott elemtől függ: ha a modellben megszüntetjük a modellelemet, akkor az az összes olyan diagramról eltűnik, amely őt mutatja.

1.2. Az UML

Az UML (Unified Modeling Language, Egységesített Modellező Nyelv) egy grafikus modellező nyelv a szoftverrendszer különböző nézeteinek modellezésére. Segítségével tervezni és dokumentálni tudjuk a szoftvereket. Az UML-ben modellek és diagramok adhatók meg, különböző nézetekben.

Az UML „csak” egy jelölésrendszer, amely független a szoftverfejlesztési módszertől. Az UML nem írja elő, hogy az egyes modelleket, illetve diagramokat mily módon és milyen sorrendben állítjuk elő.

Megjegyzés:
Az OMG (Object Modeling Group) definíciója szerint:
"Az egységesített modellező nyelv (UML) egy grafikus nyelv egy szoftver-intenzív rendszer termékeinek megjelenítésére, specifikálására, felépítésére és dokumentálására. Az UML szabványos lehetőségeket kínál a rendszer felvázolásához, beleértve a fogalmi dolgokat, mint üzleti modellezés és rendszerfunkciók, valamint a konkrét dolgokat, mint programnyelvi utasítások, adatbázis sémák és újrafelhasználható szoftverkomponensek."

Az UML a jelöléseknek gazdag választékát kínálja, mely segítségével a szoftverfejlesztés összes fázisa modellezhető. Az UML a kiterjesztési mechanizmusok (pl. sztereotípusok) segítségével egyéni jelölések s azok szemantikájának bevezetését is támogatja, hogy minden segítséget megadjon a szoftverfejlesztőknek ahhoz, hogy könnyedén modellezhessék az akár egyéni vagy különleges rendszereket is.

Ebben a könyvben modell, illetve diagram alatt elsősorban UML modellt, illetve UML diagramot értünk.

1.3. Az UML építőkövei

Az UML építőköveinek rendszerezését a következő hierarchia mutatja (a teljesség igénye nélkül):

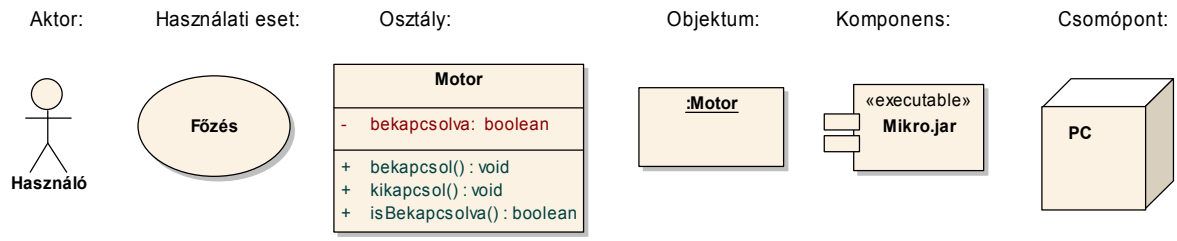
Modellelemek	Statikus dolgok	Aktor (actor) Használati eset (use case) Osztály (class) Aktív osztály (active class) Tábla (table) Objektum (object) Interfész (interface) Komponens (component) Együttműködés (collaboration) Csomópont (node)
	Dinamikus dolgok	Aktivitás (activity) Kölcsönhatás (interaction) Állapot (state) Állapot-átmenet (state transition) Folyamat (process) Esemény küldése (send) Esemény fogadása (receive) Információ (information) Döntés (decision)
	Csoportosító dolgok	Csomag (package) Alrendszer (subsystem)
	Kommentáló dolgok	Megjegyzés (remark)
Kapcsolatok	Függőség (dependency)	
	Társítás (association)	Ismeretség (acquaintance) Tartalmazás (aggregation), gyenge vagy erős.
	Általánosítás (generalization)	
	Megvalósítás (realization)	
Diagramok	Struktúramodellezés	Osztálydiagram (class diagram) Objektumdiagram (object diagram) Komponensdiagram (component diagram) Telepítési diagram (deployment diagram)
	Viselkedésmodellezés	Használatieset diagram (use case diagram) Szekvencia diagram (sequence diagram) Együttműködési diagram (collaboration diagram) Állapotdiagram (state chart diagram) Aktivitásdiagram (activity diagram)

1.4. Modellelemek

A modellelemeket részletesen mindig annál az UML diagramnál tárgyaljuk majd, amelyre az leginkább jellemző.

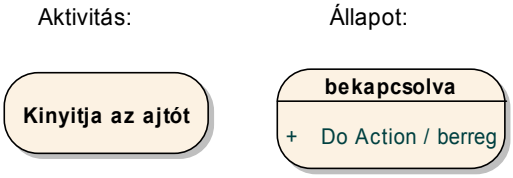
Statikus dolgok

Az UML strukturális elemei a rendszer nyugalmi állapotára jellemzők, és a rendszer struktúráját írják le. Ezek az elemek általában a rendszer leírásakor használt főnevek. Strukturális elemek például a következők:



Dinamikus dolgok

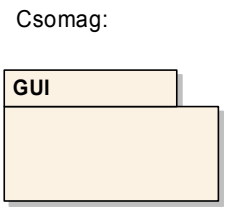
Az UML viselkedési elemei dinamikus dolgok, a rendszer mozgásának, változásainak bemutatására alkalmasak. Ezek az elemek általában a rendszer leírásakor használt igék. Viselkedési elem például az aktivitás vagy az objektum egy állapota:



Csoportosító dolgok

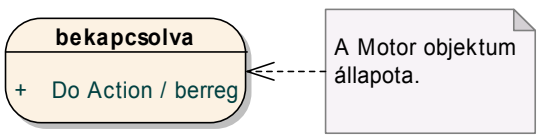
Az UML csoportosító elemei a dolgok csoportosítására, rendszerezésére való.

Csomag (package): A modellelemek csoportja. A modellelem a modellben egyértelműen azonosítható. A modellelemek csomagneveikkel minősíthetők. A diagramok csomagokat is tartalmazhatnak. A diagramon levő csomag a modell egy csomópontjának felel meg. A nézet a modell legfelső szintű csoportja.



Kommentáló dolgok

Megjegyzés (remark): Az UML-ben bármely diagramon elhelyezhető megjegyzés.



1.5. Kapcsolatok

A modellelemek között kapcsolatok definiálhatók. Az itt megadott kapcsolatok szinte minden egyes diagramon használhatók.

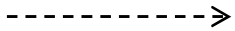
Függőség (dependency)

Logikai kapcsolat. Két dolog között függőségi kapcsolat van, ha az egyik (független) dolog változása maga után vonja a másik, (függő) dolog változását.

A függőség fajtái:

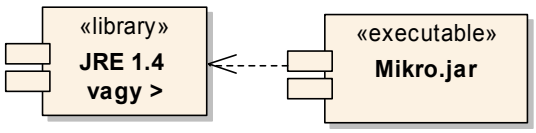
- Finomítás vagy részletezés (refinement)
- Nyomkövetés (trace)
- Tartalmazás (include)
- Kiterjesztés (extend)

Az UML-ben a függőségi kapcsolatot szaggatott nyíllal jelöljük, ahol a nyíl iránya jelzi a függőség irányát. A mutatott dolog függ a mutató dologtól. A függés lehet kétirányú is. A nyíl tartalmazhat címkét, amely leírja a függőség típusát. Elvileg bármely két dolog között lehet függőségi kapcsolat.



Például:

Objektum függ az osztályától, egyik csomag függ egy másik csomagtól, egyik komponens függ egy másik komponenstől (ábra):



Társítás (association)

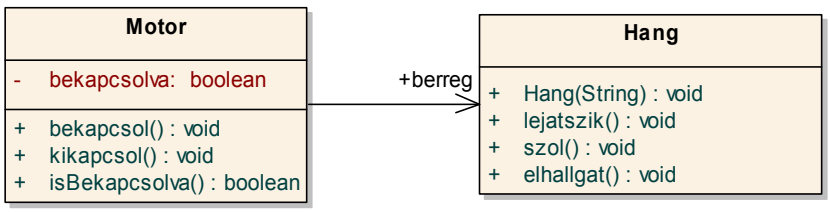
Elemek közötti közötti srukturális kapcsolat.

A társítási kapcsolat fajtái:

- Ismeretség (acquaintance)
- Tartalmazás (aggregation). Gyenge és erős tartalmazás.

Az UML-ben a tásítási kapcsolatot folytonos nyíllal jelöljük, ahol a nyíl iránya jelzi a kapcsolat irányát. Ha a nyíl hegyét nem tesszük ki, akkor a kapcsolat kétirányú. A tartalmazási kapcsolatot a nyíl elején egy tömör (erős tartalmazás) vagy üres (gyenge tartalmazás) rombusz jelzi.

Például:

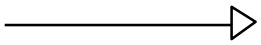


Általánosítás (generalization)

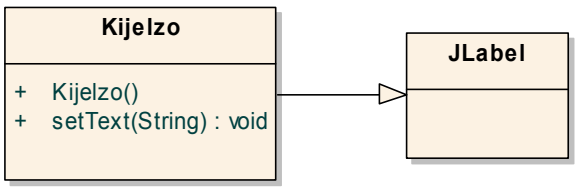
Más elnevezések: Specializáció (specialization), Öröklés (inheritance)

Osztályszerű elemek közötti strukturális kapcsolat. Általánosítási kapcsolat lehet például osztályok, aktorok, vagy használati esetek között.

Az UML-ben az általánosítási kapcsolatot folytonos, telt végű nyíllal jelöljük, ahol a nyíl iránya jelzi az általánosítás irányát.



Például:



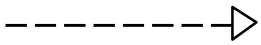
Megvalósítás (realization)

A megvalósítási kapcsolatban egy dolog megvalósít (realizálja, implementálja) egy másik dolgot. Logikai kapcsolat, mely az általánosítás és függőség egy keveréke. A kapcsolat csak osztályszerű elemek között lehetséges.

Megvalósítási kapcsolat a következő dolgok között lehet például:

- Interfész és osztály (az osztály megvalósítja az interfészt)
- Osztály és komponens (a komponens megvalósítja az osztályt)
- Folyamat és használati eset (a használati eset megvalósítja a folyamatot)
- Használati eset és együttműködés (az együttműködés megvalósítja a használati esetet)

Az UML-ben a megvalósítási kapcsolatot szaggatott telt végű (öröklési) nyíllal jelöljük, ahol a nyíl iránya jelzi a megvalósítás (függőség) irányát.



1.6. Diagramok

A könyben 11 féle diagramot tárgyalunk, az UML modellek ezeket a diagramokat tartalmazhatják (lásd ábra). Az UML diagramokat a modellezés fajtája szerint három csoportba oszhatjuk:

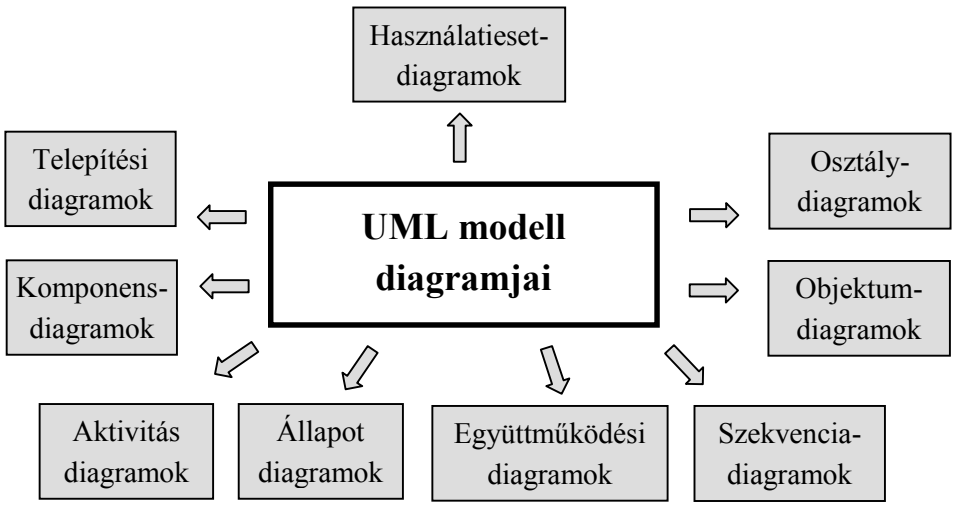
Struktúramodellezés (a rendszer struktúráját ábrázoló diagramok):

- **Osztálydiagram** (class diagram): Megadja a rendszer osztályait, és az azok közötti társítási és öröklési kapcsolatokat. Az adatbázisdiagram (database diagram) egy speciális osztálydiagram, amely megadja a rendszer perzisztens adatainak szerkezetét.
- **Objektumdiagram** (object diagram): Megadja a rendszer objektumait, és az azok közötti kapcsolatokat. Az osztálydiagram egy „pillanatfelvétele”.
- **Komponensdiagram** (component diagram): Megadja a szoftver fizikai felépítését, vagyis azt, hogy a tervezési elemek szoftver elemekre való leképezését.
- **Telepítési diagram** (deployment diagram): Megadja, hogy milyen szoftver elemeket milyen hardverre telepítünk.

Viselkedésmodellezés (a rendszer viselkedését ábrázoló diagramok):

- **Használatieset diagram** (use case diagram). Megadja, hogy a felhasználó mire tudja használni a rendszert. Aktorokat, használati eseteket és azok kapcsolatait ábrázoló diagram. Funkcionális követelményeket ábrázoló diagram.

- **Szekvenciadiagram** (sequence diagram): Aktorokat, objektumokat és az azok közötti kapcsolatokat, kölcsönhatásokat (üzeneteket) ábrázoló diagram. A szekvenciadiagramot és az együttműködési diagramot együttesen interakciódiagramoknak nevezzük. A szekvenciadiagram olyan interakciódiagram, mely az idő múlására helyezi a hangsúlyt.
- **Együttműködési diagram** (collaboration diagram): Megadja a rendszer objektumait, az azok közötti kapcsolatokat és üzeneteket. Az együttműködési diagram az osztálydiagram egy „pillanatfelvétele". Az együttműködési diagram a szekvenciadiagram egy más formája – olyan interakciódiagram, mely az objektumok közötti kapcsolatra helyezi a hangsúlyt.
- **Állapotdiagram** (state diagram): Egy adott osztály vagy alrendszer állapotváltozásait írja le.
- **Aktivitásdiagram** (activity diagram): Leír egy folyamatot (tevékenységek egymásutánját). Az üzleti folyamat diagram egy speciális aktivitásdiagram, mely leírja a rendszert körülvevő folyamatokat, illetve azt a környezetet, amelybe a rendszert el kell helyezni.



UML diagramok

Minden egyes diagramnak más a célja, azok a rendszert más-más szempontból világítják meg. Elvileg bármelyik modellhez bármelyik diagram felhasználható. Az egyes diagramokat a következő fejezetek egyenként fogják tárgyalni az őt alkotó jellegzetes modellelemekkel együtt.

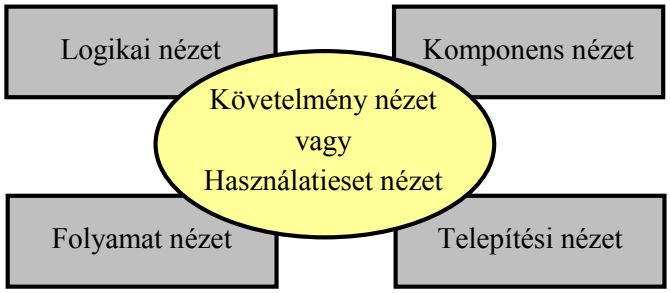
1.7. Az UML modelljei

UML modellnek nevezzük az UML jelölésrendszerével megadott modellt. Az UML modell elemeit UML dolgoknak is mondják. Az UML diagramokat UML dolgok és az azok közötti kapcsolatok alkotják. Az UML diagram is UML dolog.

Az UML modelljei a következők:

- **Üzleti modell:** Bemutatja a rendszer üzleti folyamatait.
- **Követelménymodell:** Megadja a rendszerrel szemben támasztott követelményeket. Megadja, hogy a rendszert mire akarják majd használni (funkcionális követelmények), és hogy milyen egyéb feltételeknek kell tell teljesülniük (nemfunkcionális követelmények).
- **Tervezési modell:** Megoldási útmutató.
- **Implementációs modell:** Kivitelezési útmutató.
- **Teszt modell:** A rendszer működésének kipróbálására vonatkozó útmutató.

1.8. Az UML nézetei



Az UML nézetei

A modell nézetei a modell legfelső szintű csomagjai. Az UML által definiált nézetek a következők:

- **Követelmény nézet** (Requirements View) vagy **Használatieset nézet** (Use Case View): A rendszer modellezése a felhasználó, megrendelő szemszögéből. E nézet segítségével megfogalmazhatók a rendszerrel szemben támasztott felhasználói követelmények. Megadható, hogy kik és mire akarják használni a rendszert. Itt írjuk le a projekt funkcionális és nemfunkcionális követelményeit. Ez a nézet a fejlesztés kezdeti fázisaiban lényegében kialakul, és végigkíséri a teljes fejlesztést.
- **Dinamikus nézet** (Dynamic View) vagy **Folyamat nézet** (Process View): A rendszer működését, folyamatát, dinamikáját ábrázolja. Elsősorban interakció, aktivitás és állapotdiagramokat tartalmaz.
- **Logikai nézet** (Logical View) vagy **Tervezési nézet** (Design View): Ebben a nézetben adjuk meg a szakterületi modellt és a részletes osztálymodellt. Itt adjuk meg a felhasználói interfész, az adatbázis és az alkalmazáslogika osztályait, szükség szerint csomagokba sorolva. Ez a programterv lényegi része, ez alapján történik a kódolás, illetve a kódgenerálás.
- **Komponens nézet** (Component View) vagy **Implementációs nézet** (Implementation View): A rendszert alkotó fizikai komponensek, szoftverelemek modellezése. Itt adható meg a logikai nézet osztályainak forráskomponensekhez való hozzárendelése, valamint a forráskódok hozzárendelése futtatható komponensekhez. Az implementációs nézet a rendszer állományait modellezi. A nézet jellemző diagramja a komponensdiagram.
- **Telepítési nézet** (Deployment View): A rendszer topológiájának modellezése. Leírja a rendszer szoftver elemeinek a hardver elemekhez való rendelését. Itt adható meg, hogy milyen számítógépen milyen szoftver fut. A nézet jellemző diagramja a telepítési diagram.

1.9. Az UML általános mechanizmusai

(Még hiányos)

Az UML általános mechanizmusai mintákat adnak arra vonatkozóan, hogy hogyan lehet az egyes építőköveket bővebb ismeretekkel felruházni, szemantikájukat bővíteni. Az általános mechanizmusok használatával lehetőség nyílik sajátos felhasználói építőkövek létrehozására.

Specifikációk (specifications)

A dolgok szöveges leírása. Az UML-ben minden egyes dologhoz tartozik egy specifikáció, melyben leírható a dolog szintakszisa, illetve szemantikája.

Dekorációk (adornments)

A dolog grafikus reprezentációja alapértelmezésben csak a leglényegesebb jelöléseket tartalmazza. A dekoráció alkalmas részletek, kiegészítések megadására.

Például:

- Láthatóságok: + - #
- Absztrakt osztály: ferde szedés

Általános felosztások (common divisions)

Osztály-objektum

A dolgok között vannak osztályszerű elemek, és vannak példányszerű elemek. Az osztály egy absztrakció (leírás, minta), az objektum (példány) pedig az absztrakció egy konkrét megnyilvánulása. A két dolog jelölése abban különbözik, hogy az objektumot aláhúzzuk. Például:

Osztály	Objektum
Használati eset	Forgatókönyv
Osztályok közötti kapcsolat	Objektumok közötti kapcsolat
Csomópont	Csomópont példány
Komponens	Komponens példány

Interfész-megvalósítás

A dolgok között vannak interfész-szerű elemek és azok megvalósításai (implementációi). Például:

Interfész	Osztály
Üzenet	Metódus
Használati eset	Együttműködés

Kiterjesztési mechanizmusok (extension mechanism)

Sztereotípus (stereotype)

Dolgok körének bővítése. Mivel az UML nem képes a világ összes bonyolult rendszerének modellezésére. A sztereotípus segítségével azz UML elemei bővíthetők.

Megjelölt értékek (tagged values)

Megkötések (constraints)

A dolgok szemantikájának bővítése.

Általános kiterjesztési mechanizmusok:

- Megszorítások és megjegyzések
- Elemek tulajdonságai
- Sztereotípusok

2. CASE eszköz – Enterprise Architect

Egy szoftverfejlesztő CASE eszköz segítségével a szoftvert modellezni tudjuk. Ez a fejezet az Enterprise Architect (rövidítve: EA) alapfokú használatát és legfontosabb elemeit tárgyalja. Bemutatjuk a képernyő felépítését, a modellelemek tárolásának filozófiáját és a legfontosabb technikai lehetőségeket. Az egyes modellelemeket nem itt mutatjuk be részletesen – a CASE eszközt a könyvben végig használni fogjuk, a különböző lehetőségeket a megfelelő részekben tárgyaljuk.

2.1. A szoftver modellezése CASE eszközzel

Egy szoftverfejlesztő CASE eszköz célja: Szoftver modellezése annak elkészítése, dokumentálása és továbbfejlesztése céljából.

Egy szoftverfejlsztő CASE eszköz elvárható tulajdonságai

- Modellezés több absztrakciós szinten, több nézetben
- Modellelemek közös tárháza (repository)
- Rajzolás
- Navigáció, átjárhatóság
- Többfelhasználós támogatás
- Kódgenerálás és kódvisszafejtés
- Más eszközökkel való együttműködés
- Más modellek exportálása, importálása
- Dokumentációkészítés

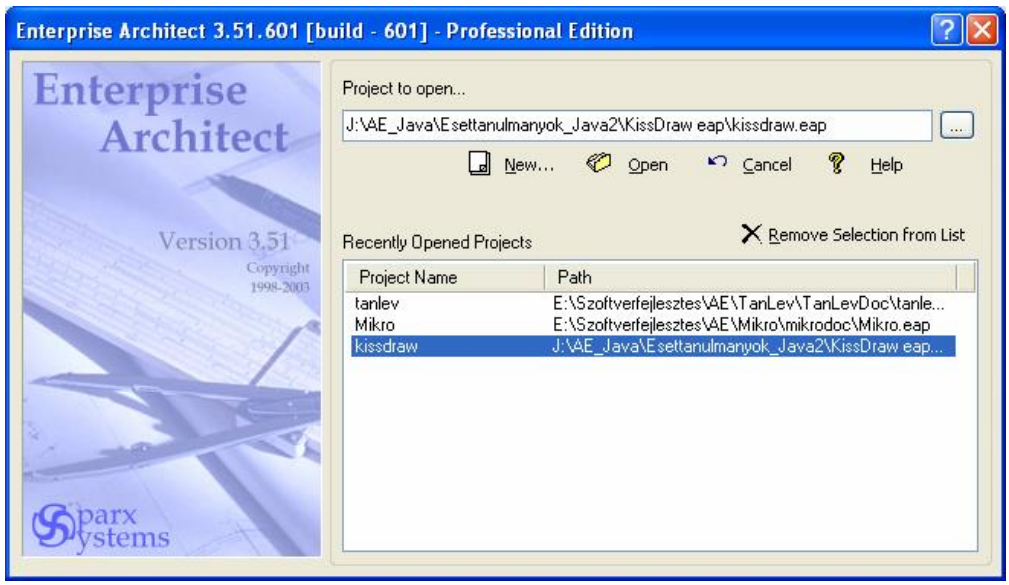
Az Enterprise Architect egy szoftverfejlesztő CASE eszköz, melyet a Sparx Systems fejlesztett ki (http://www.sparxsystems.com.au).

2.2. Enterprise Architect projekt

A **projekt** a modellezés fizikai egysége. Projektet általában egy adott szoftver fejlesztéséhez alakítunk ki és használunk.

Az Enterprise Architect egyszerre csak egy projektet képes kezelni. Az EA a projektet egy EAP kiterjesztésű fájlban tárolja (Enterprise Architect Project).

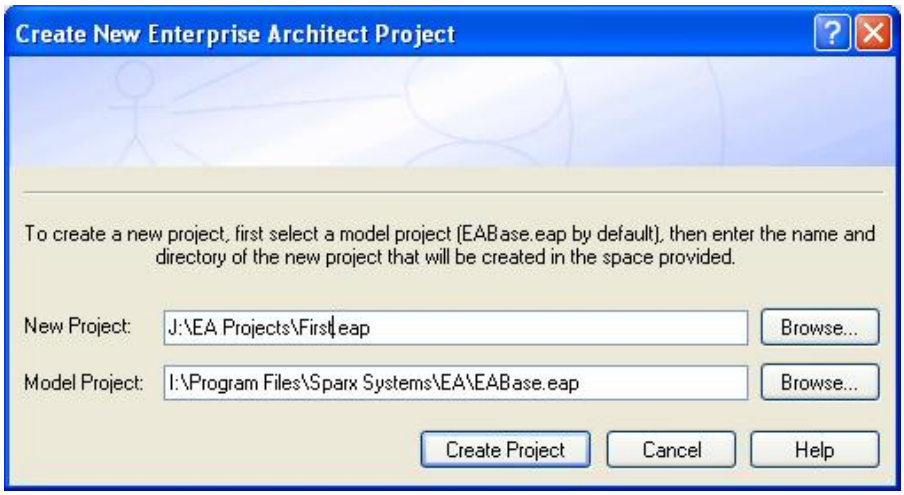
Projekt betöltése



A betöltendő projekt kiválasztható a lemezen való böngészéssel (Project to open...), de újranyitható valamelyik legutoljára betöltött projekt is (Recently Opened Projects).

Projekt létrehozása

Az EA-ban új projektet (New Project) egy régi projekt másolásával hozhatunk létre. A másolandó projekt (minta, sablon) a modell projekt (Model Project).



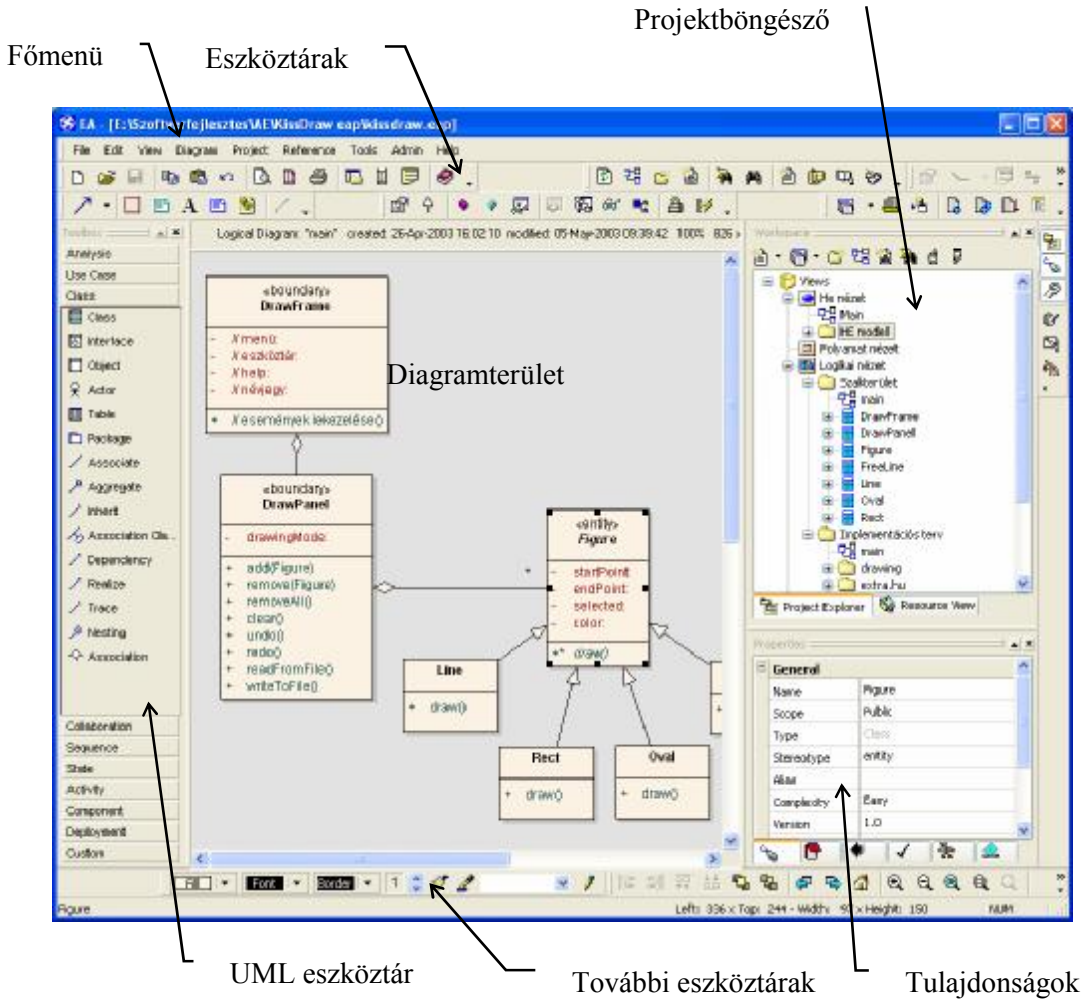
A képernyő felépítése

Az Enterprise Architect képernyője a következő fő részekből áll (lásd ábra):

- Főmenü (Main menu). Elemei a szokásosak: File, Edit, View, Diagram stb.
- Eszköztárak (Toolbars). Itt találhatók csoportosítva a főmenü fontosabb elemei. Az eszköztár csoportjait külön pont tárgyalja majd.
- Projektböngésző (Project browser): A projektböngészőben a projektfa elemeit böngészhetjük. A projektböngészővel a következő pont foglalkozik.
- Tulajdonságok (Properties): Itt jelennek meg a projektböngészőben vagy a diagramterületen legutoljára kijelölt elem legfontosabb tulajdonságai, több fülre elosztva. Tulajdonságok például:

név (Name), láthatóság (Scope), típus (Type), sztereotípus (StereoType), bonyolultság (Complexity), verzió (Version), az elem szerzője (Author), megjegyzés (Notes) stb.

- Diagramterület (Diagram area): Ez az alkalmazás fő munkaterülete, a képernyő közepén helyezkedik el. Itt jelenik meg a projektfán legutoljára kijelölt diagram. Egyszerre csak egy diagram látható, illetve szerkeszthető. Egy diagramra bármely projektelem rátehető.
- UML eszköztár (UML toolbar): A képernyő bal odalán helyezkedik el. Itt vannak a modellelemek, melyek a rátehetők az aktuális diagramterületre. A modellelemek csoportosítva vannak: egy adott diagramra jellemző modellelemek egy csoportban vannak.



Az Enterprise Architect képernyőfelépítése

2.3. Pojektfa, projektböngésző

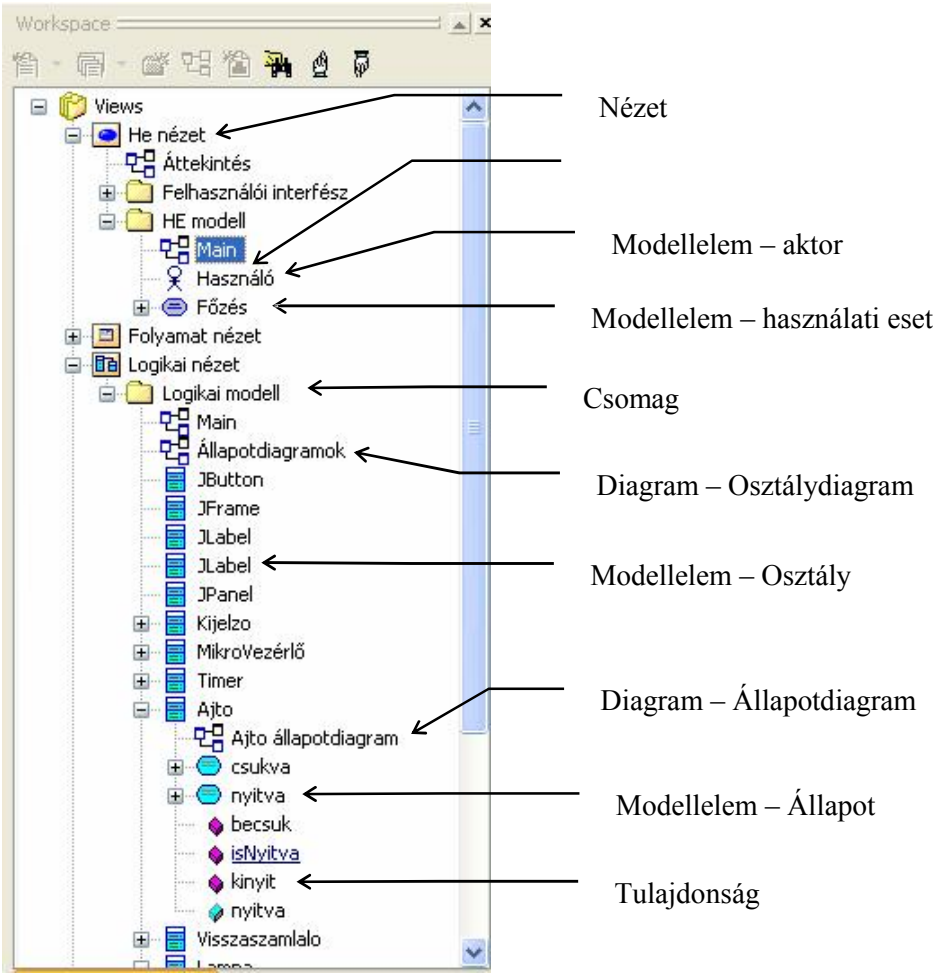
Az EA projekt legfelső szinten nézeteket tartalmaz. Ezek a nézetek lehetnek lehetnek egyetlen modell nézetei, de a nézetek alá nyugodtan betehetünk különböző modelleket is. A modellelemek az egész projektben egyértelműen azonosíthatók.

A projektböngésző (projektfa) egy **tárház** (repository), mely fastruktúrában tartalmazza a projekt összes modellelemét (aktorokat, használati eseteket, osztályokat, objektumokat stb.) és diagramjait.

A **projektfa** csomópontjai a csomagok. A projektfa legfelső szintű csomagjai a **nézetek** (Views). Itt megadhatók az UML szabványos nézetei, de egyéni nézetek is beiktathatók.

A projektböngésző a modellelemek elsődleges helye. A modellelemek a projekten belül egyértelműen azonosíthatók. Minden egyes modellelem a projektfa egy egyértelmű csomópontjában (csomagjában) van.

Az ábrán a Mikro.eap projektfáját láthatjuk.



Projektfa – Mikro

A nézetek nevei alapértelmezésben angolok, de a projektfa bármely elemét át lehet nevezni, így a nézeteket is át lehet írni magyarra. Az Enterprise Architect-ben lehetőség van további, felhasználó által definiált nézetek ábrázolására (pl. Custom View, saját nézet). Saját nézetben például meg lehet adni egy komplex felhasználói interfész tervet.

Egy nézet alatt a következő elemek helyezhetők el:

- Modellelemek (az UML eszköztár elemei). A modellelemek alatt helyezkednek el azok tulajdonságai, de a modellelem alatt elhelyezhetők újabb diagramok is.
- Diagramok
- További csomagok

A fa tetszőlegesen alakítható, beszúrhatók új elemek, már meglévőek törölhetők, átnevezhetők. Az elemek átrendezhetők "fogd és vidd" technikával.

Egy csomag jellemző felépítése:

```
Diagram
Diagram
...
Csomag
Csomag
...
Modellelem
  Tulajdonság
  Tulajdonság
  ...
  Diagram
  Diagram
  ...
Modellelem
...
```

A csomagban további diagramok, modellelemek és diagramok lehetnek. Az egymásba ágyazás elvileg korlátlan mélységű.

 PÉLDA

- a *Főzés* egy használati eset –a Használati eset nézet eleme;
- az *Ajtó* egy osztály – a Logikai nézet eleme;
- az *Ajtó* osztály alatt van annak Állapotdiagramja és az állapotdiagram modellelemei.

2.4. Diagram

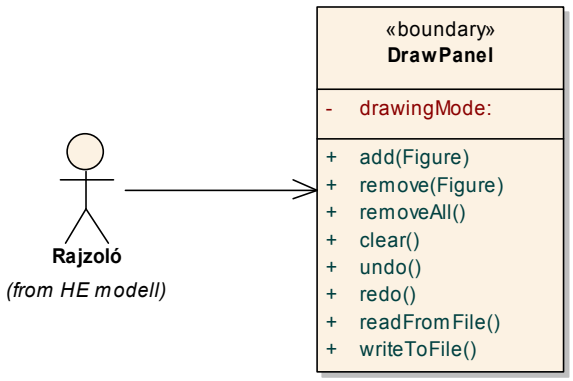
Az egyes nézetekben létrehozható fontosabb modellelemeket és diagramokat a következő táblázat mutatja:

	Modellelemek	**Diagramok**
Használatieset (követelmény) nézet	csomag (package) aktor (actor) használati eset (use case)	használati eset (use case) osztály (class) együttműködési (collaboration) szekvencia (sequence) állapot (statechart) aktivitás (activity)
Logikai nézet	csomag (package) osztály (class) objektum (object) interfész (interface)	használati eset (use case) osztály (class) együttműködés (collaboration) szekvencia (sequence) állapot (statechart) aktivitás (activity)
Komponens nézet	csomag (package) komponens (component)	komponens (component)
Telepítési nézet	csomag (package) csomópont (node)	telepítési (deployment)

Szabályok:

- A diagramok között lehet átfedés.

- Egy modellelem akárhány diagramon szerepelhet.
- Egy modellelemet nem kell feltétlenül diagramon szerepeltetni.
- Egy modellelemet egy konkrét nézetben hozunk részre, ahhoz tartozik.
- Egy diagramon idegen (más nézethez tartozó) modellelem is szerepelhet. Ilyenkor a modellelem nevében megjelenik a from minősítés. A lenti ábrán például a HE nézetben létrehozott aktort megjelenik a logikai nézetben.
- Ha egy modellelemet a projektböngészőből (projektfáról) törlünk, akkor a modellelem összes megjelenése törlődik (az összes diagramról törlődik). Ez fordítva nem igaz: ha egy diagramról törlünk egy elemet, attól még a modellelem továbbra is létezik.



CASE

Modell elem törlése:

- Aktuálissá tesszük a modell elemet a böngészőben
- Jobb egérrel láthatóvá tesszük a felbukkanó menüt.
- Kiválasztjuk a Delete menüpontot

Elem törlése a diagramról:

- Aktuálissá tesszük a modell elemet a grafikonon
- Leütjük a Delete gombot.

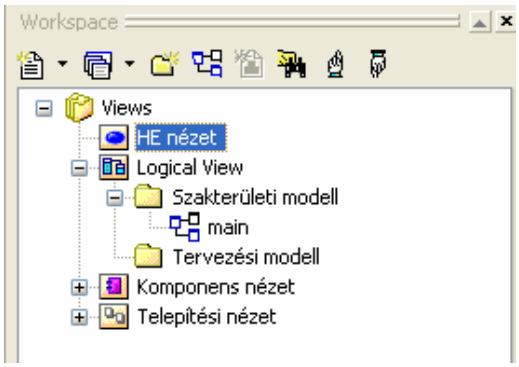
2.5. Sablon készítése

Készítsünk egy egyszerű sablont modelljeink elkészítéséhez!

CASE

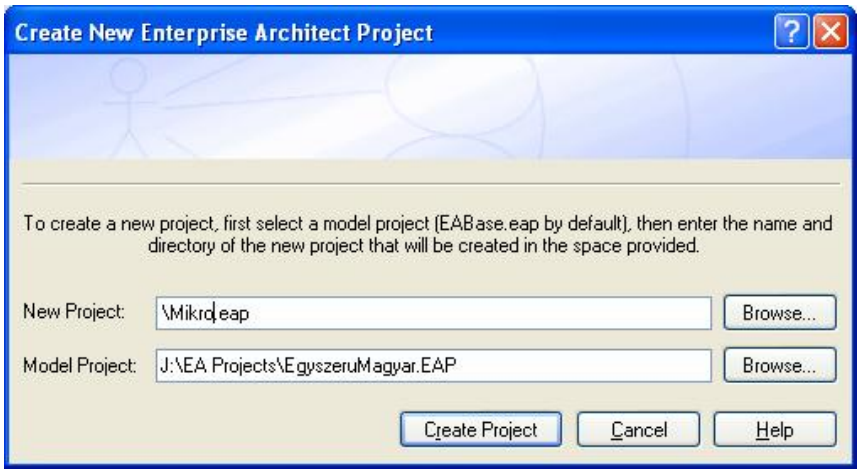
Indítsuk el az Enterprise Architect szoftvert! Hozzon létre egy új projektet a gyárilag adott EASample.EAP projekt alapján!

Alakítsuk át a nézeteket az ábrán látható módon! A feladatot a megfelelő csomagok törlésével és átnevezésével lehet megoldani.



Mentse el a fájlt EgyszeruMagyar.EAP néven!

Legközelebb egy új projekt létrehozásánál ezt a mintát használjuk!



2.6. Eszköztárak (Toolbars)

Itt találhatók a főmenü fontosabb elemei, csoportosítva:

- Default toolbar: File, Edit, Help stb. legfontosabb pontjai
- Project: Projektelemek kialakítása, objektumok keresése, fogalomtár szerkesztése stb.
- Code generation: Kódgenerálással kapcsolatos funkciók, mint adott nyelvű forrásprogramok generálása a megadott osztályokból, adott fossáskód visszafejtése osztályokká, szinkronizálás, stb.
- UML elements: Az UML eszköztár leggyakrabban használatos elemei, mint megjegyzés, kapcsolatok, link stb.
- Diagram: az aktuális diagram elrendezésével és bejárásával kapcsolatos lehetőségek, mint előző/következő, igazítás, színek, előre/hátra, nagyítás/ kicsinyítés ...
- Element: Az aktuális diagramelem tulajdonságainak beállítása: specifikáció, adatok, metódusok, kapcsolatok, az elem keresése a projektfán stb.
- Connector: A diagram két eleme közti vonal szemantikájának és stílusának megadása.
- Color tool: Az egyes diagramelemek színeinek megadása.

Az egyes csoportok és azok elemei értelemszerűen akkor aktívak, ha az alkalmazás kérdéses részei vannak fókuszban.

3. Üzleti folyamat diagram

Az üzleti folyamatdiagram egy speciális aktivitásdiagram. A fejlesztés kezdeti szakaszában fel kell mérnünk, hogy az adott szakterületen, „az üzletben” milyen folyamatok zajlanak. Sok esetben egy szoftver használatát más kézi vagy gépi tevékenységek egészítik ki, például egy kinyomtatott számlát iktatni kell. Az üzleti folyamatok modellezésével világosan megadható, hogy mi az adott szakterületen működő egy vagy több szoftver, illetve szoftver-rész feladata, és hogyan illeszkednek azok környezetükbe.

3.1. Az üzleti folyamatdiagram szerepe a modellezésben

Az **üzleti folyamatmodell** (Busines Process Model, BPM) célja egy szervezet folyamatainak feltárása a szakterület, valamint a szervezetben működő tevékenységek és szoftverek megértése érdekében. Az üzleti folyamatmodellezés segítségével definiálhatjuk, illetve megérthetjük egy adott szoftver bővebb környezetét, funkcióit és határait. Az üzleti folyamadmodell bővebb, mint az abba beleágyazott szoftver(ek). Az üzleti folyamatmodell az üzlet valódi életét mutatja be; tartalmazhat olyan üzleti tevékenységeket is, melyek a szoftvereket körülveszik és összekapcsolják kézi, gépi vagy más módon.

Az üzleti folyamatmodell diagramjai az **üzleti folyamatdiagram**ok. Az üzleti folyamatdiagram egy gráf, mely a teljes üzleti folyamat egy kiragadott részét ábrázolja.

Megjegyzés:
Az üzleti folyamatdiagram egy speciális aktivitásdiagram az üzleti folyamatok modellezésére, melyet először Hans-Erik Erikkson és Magnus Penker definiáltak.

3.2. A diagram elemei

Az üzleti folyamatdiagram elemei a következők:

- Folyamat (benne tevékenység)
- Bemeneti objektum vagy információ
- Kimeneti objektum vagy információ
- Küldött esemény
- Fogadott esemény

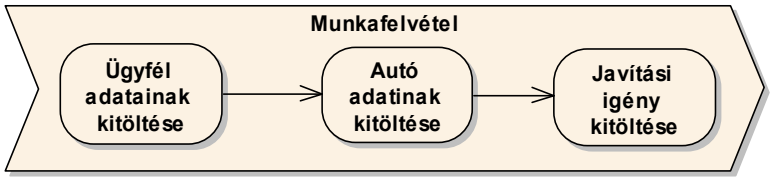
Folyamat (üzleti folyamat)

A folyamat egy speciális tevékenység, mely üzleti folyamatot ír le. A folyamat a folyamatdiagram központi eleme.

Az üzleti folyamat tevékenységek gyűjteménye, melyek együttes célja valamely kimenet előállítása a fogyasztó (megrendelő) vagy a piac számára. A folyamat a szervezetben folyó munkák időben és térben való ábrázolása. A folyamatnak mindig van egy jól meghatározott kezdete és vége. A folymatnak lehetnek bemenetei, kimenetei, céljai stb.

A folyamat jele:

A folyamat tevékenységek összessége (folyamata). Haladási irány: balról jobbra. A folyamat modellelemen belül meadhatók a folyamat belső tevékenységei:



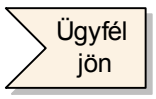
A folyamat bal oldalán általában egy küldő esemény van (ez indítja el a folyamatot), jobb oldalán pedig egy kimenet (ezt produkálja a folyamat, vagy ez a célja a folyamatnak):

Esemény

Az esemény egy történés, egy értesítés vagy egy időpont eljövetele. Az esemény elindítja az üzleti folyamatot. Az eseményt az üzleti folyamat felhasználhatja, átalakíthatja vagy továbbíthatja.

Az esemény lehet küldött esemény (send) vagy fogadott esemény (receive).

A fogadott esemény jele:



A küldött esemény jele:

A folyamat által indukált, küldött eseményt egy másik folyamat fogadhatja. Ebben az esetben bármelyik szimbólum használható.

Bemeneti objektumok

Az üzleti folyamat végrehajtásához általában szükség van bemeneti objektumokra – a folyamat ezekből áplálkozik. A bemeneti objektum bármi lehet, például:

- perzisztens adattár (adatbázis);
- átmeneti adattár (adatok a memóriában);
- fizikai dolog (pl. bőr, mint nyersanyag);
- információ (pl. Internet info, felhasználó által szolgáltatott adatok);
- erőforrás (pl. rendelés).

Az információt a folyamat használja, de nem változtatja meg. Az erőforrást a folyamat „elfogyasztja”, vagyis feldolgozás közben az erőforrás megváltozik, elfogy.

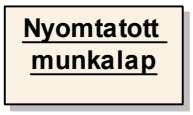
A bemeneti objektum sztereotípusát akár az objektumon, akár a bemeneti kapcsolaton feltüntethetjük.

Kimeneti objektumok

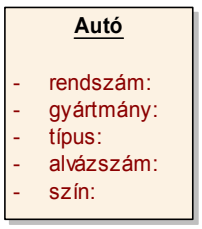
Az üzleti folyamatnak vannak kimeneteti objektumai – a folyamat ezeket produkálja, ez jön ki belőle. A kimeneti objektum elvileg bármi lehet, például:

- perzisztens adattár (adatbázis);
- nyomtatott lista;
- fizikai dolog (pl. cipő, mint gyártmány);
- információ (pl. egy grafikon, egy szám vagy egy képernyőlista);

A bemeneti és kimeneti objektumok jele a következő:



Az objektumoknak lehetnek adatai (vagy akár metódusai) is:



Az információobjektum jele egy rombusz. Adatokat ebbe is írhatunk:

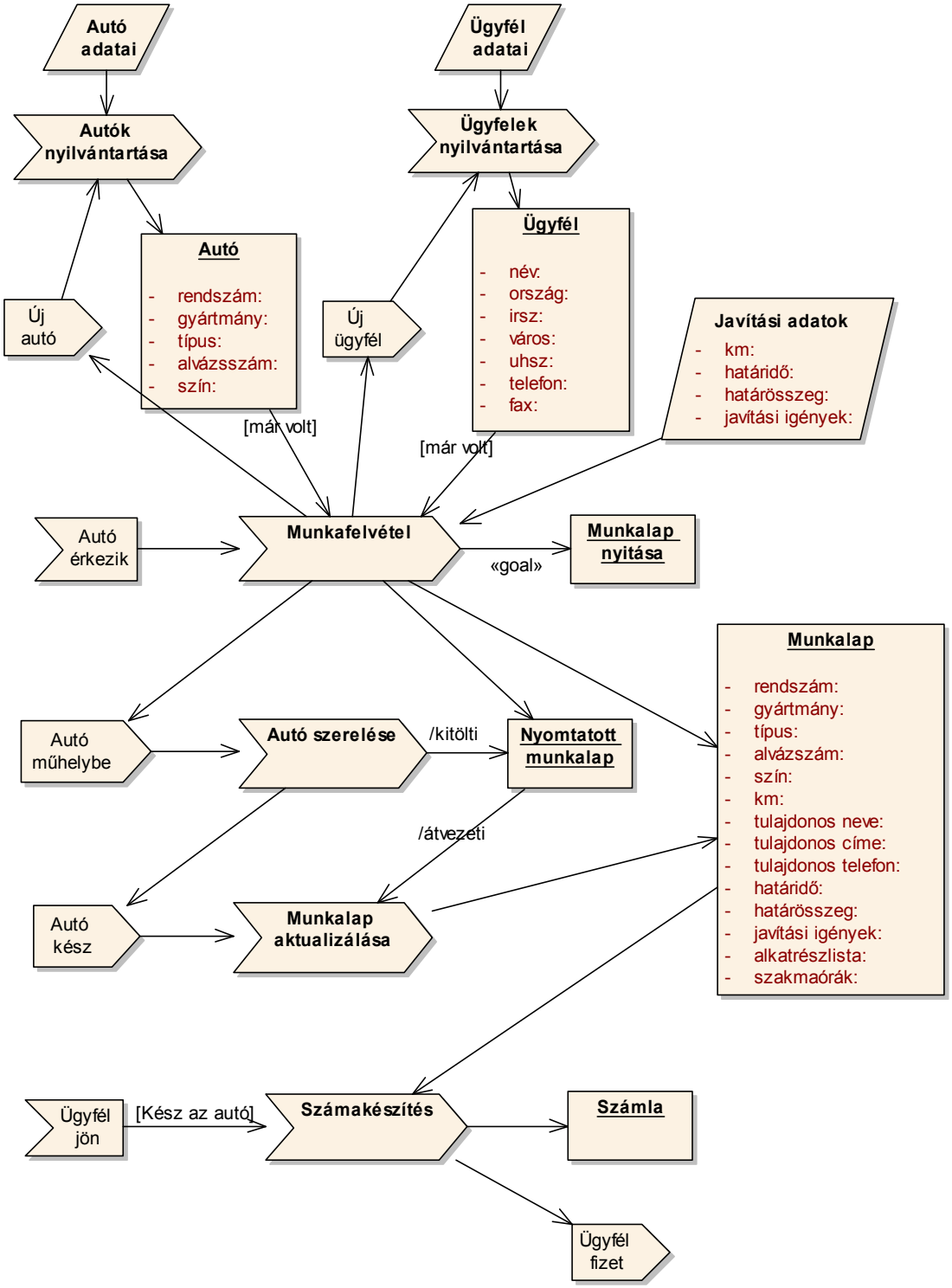
Minta diagram

Az ábrán a Szerviz alkalmazás üzleti folyamatdiagramja látható. A legelső munkafolyamat a "Munkafelvétel", ezt követi az "Autó szerelése", majd a "Munkalap aktualizálása", végül a "Számlakészítés".

"Munkafelvétel" folyamat

Bemeneti események, objektumok:

- Autó érkezik: Esemény, ez indítja el a folyamatot.
- Autó adatai. A munkalap kitöltéséhez van rá szükség. Ha az autó adatai szerepelnek a nyilvántartásban (már volt itt), akkor ezt az információt használjuk. Ha új autóról van szó, akkor megindul az "Autó nyilvántartása" folyamat.
- Ügyfél adatai. A munkalap kitöltéséhez van rá szükség. Ha az ügyfél adatai szerepelnek a nyilvántartásban (már volt itt), akkor ezt az információt használjuk (ha volt autó, akkor annak az ügyfelét vesszük). Ha új ügyfélről van szó, akkor megindul az "Ügyfél nyilvántartása" folyamat.
- Javítási adatok: Az ügyfél bemondja az autóval kapcsolatos panaszait, javítási kívánságait.



A Szerviz alkalmazás üzleti folyamatdiagramja

Kimeneti események, objektumok:

- Munkalap nyitása: Ez a folyamat célja.
- Munkalap: Elkélszül egy munkalap a kezdeti adatokkal.
- Nyomtatott munkalap: A munkalapot kinyomtatják, ezt az "Autó szerelése" folyamat fogja használni: a konkrét szereléseket a szerelők kézileg rávezetik.
- Autó műhelybe: Esemény. Az autót beviszik a műhelybe. Ez az esemény egyben az "Autó szerelése" folyamat indító eseménye is.

"Autó szerelése" folyamat

Ez a folyamat nem része a készülő számítógépes szoftvernek.

Bemeneti események, objektumok:

- Autó műhelybe: Esemény, ez indítja el a folyamatot.

Kimeneti események, objektumok:

- Autó kész: Esemény, a szerelés eredménye.
- Nyomtatott munkalap: Ez egy papír, melyet a szerelő kitölt, vagyis rávezeti a konkrét javításokat.

"Munkalap aktualizásása" folyamat

Az "Autó kész" esemény elindítja a folyamatot. A folyamat felhasználja a nyomtatott munkalapot, mely alapján módosítja a Munkalap objektumot.

"Számlakészítés" munkafolyamat

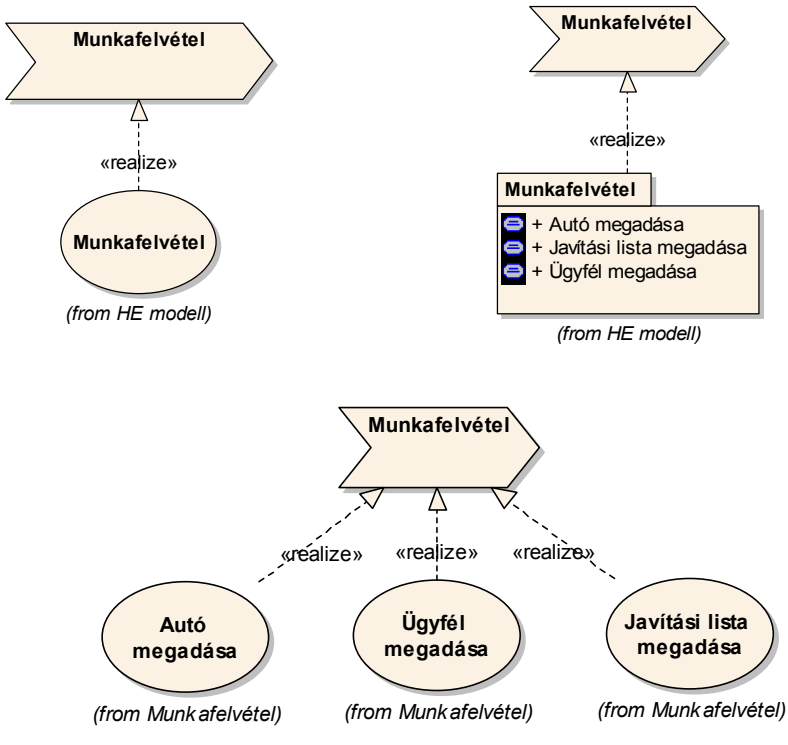
Az "Ügyfél jön" esemény elindítja a folyamatot. Ha az autó kész, akkor a Munkalap alapján elkészül a számla. Az ügyfél fizet, de ez nem része a szoftvernek.

Üzleti folyamat – használati eset

Az üzleti folyamatokat majd a használati esetek fogják megvalósítani. Az implementáció fázisában az üzleti folyamatokhoz használati eseteket vagy azok egy csoportját (csomagját) lehet kapcsolni. A kapcsolat megvalósítási (realization).

Az üzleti folyamatmodell végső, implementációs változatáról le lehet olvasni, hogy mely folyamatok nincsenek megvalósítva az adott szoftverben. Ha egy folyamathoz egyáltalán nem kapcsolódik használati eset, akkor az nem része a szofternek (lehet például egy kézi folyamat).

Tegyük fel, hogy a "Munkafelvétel" folyamatot a "Munkafelvétel" használati eset valósítja meg, melynek három alesete van. A Munkafelvétel folyamat realizálását többféleképpen is megadhatjuk:



4. Követelménydiagram

4.1. Követelménymodell, követelménydiagram

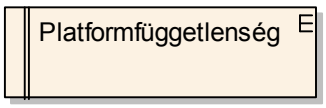
A **követelménymodell** tartalmazza a rendszerrel szemben támasztott összes követelményt, elvárást. A modellelemek a követelmények, melyek között társítási és öröklési kapcsolatok lehetségesek. Ezzel a követelmények hierarchiába szervezhetők. A **követelménydiagram** egy gráf, mely követelményeket és azok közötti kapcsolatokat ábrázolnak.

4.2. Követelmény

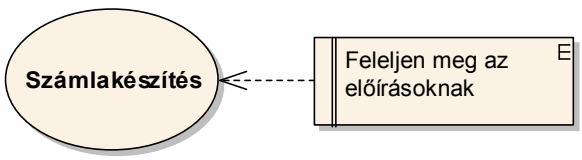
Követelménynek nevezzük a szoftverrendszerrel szemben támasztott igényt. Vannak funkcionális és nem funkcionális követelmények:

- funkcionális követelmény: mit kell végrehajtania a programnak;
- nemfunkcionális követelmény: milyen feltételeknek kell eleget tennie a programnak. A nemfunkcionális követelmények a következő dolgokkal lehetnek például kapcsolatosak: képernyőn való megjelenés, nyomtatás, teljesítmény, tesztelés.

A követelmény jele:



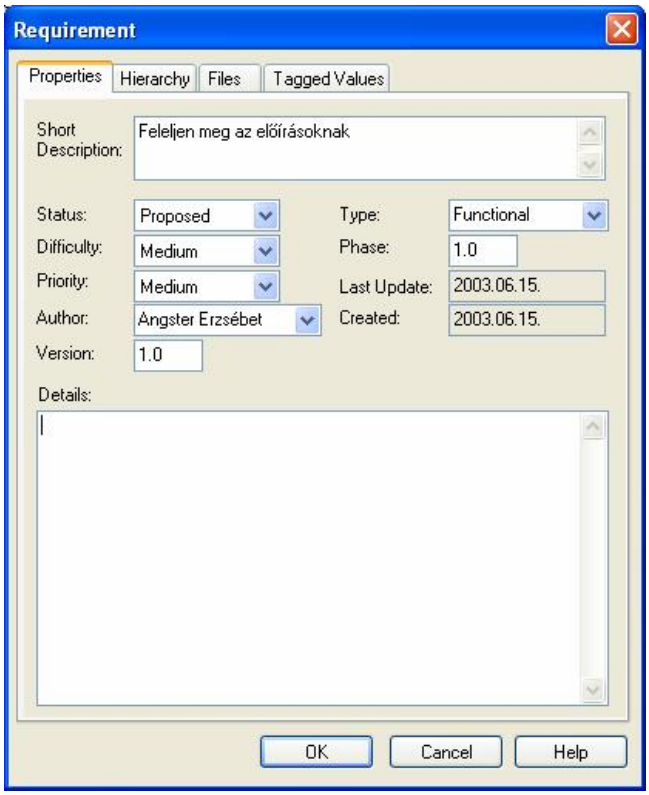
A követelmények a megfelelő diagram megfelelő modelleleméhez hozzákapcsolható, jelezve ezzel, hogy a modellelem e követelmény teljesítését szolgálja, illetve implementálja:



Egy követelmény betehető bármely modellelem felelősségei közé, és ki is vehető onnan.

A követelménynek többek között a következő tulajdonságai lehetnek:

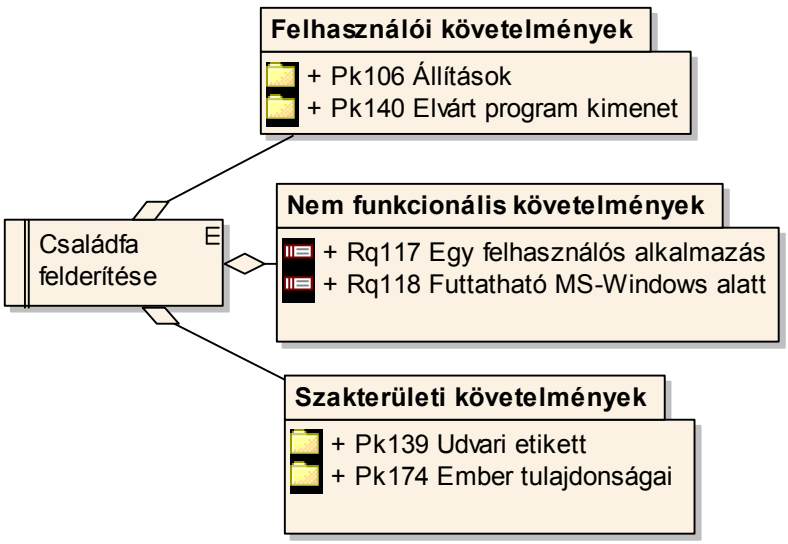
- státusz: előterjesztett / jóváhagyott / kötelező / érvényesített / megvalósított;
- felelős: aki nevéhez fűződik;
- mikor hozták létre és mikor módosították utoljára;
- nehézség;
- prioritás.



A követelményeket típusokba szokás sorolni a könnyebb áttekinthetőség kedvéért. Jellegzetes követelménytípusok: funkcionális (functional), megjelenés (diplay), teljesítmény (performance), nyomtatás (printing), riport (report), tesztelés (testing) és érvényesítés (validate).

4.3. Minta követelménydiagram

A következő minta Zsembery Ágoston Családfa programjából való:



4.4. A követelmények megvalósítása

A funkcionális követelményeket használati esetek realizálják. A használati eseteket pedig képernyőablakok, alkalmazáslogika és adatkezelő osztályok realizálják.

5. Használati-eset diagram

A használatieset (HE) diagram a rendszer viselkedésének egy kiragadott részét írja le külső aktorok szemszögéből. A HE diagram elemei: aktorok, használati esetek, valamint az ezek közötti kapcsolatok.

5.1. Aktor (szereplő)

Egy aktor üzeneteken keresztül kommunikál a rendszerrel. Az aktor üzenetet küld a rendszernek, a rendszer pedig a használati eset végrehajtása közben üzeneteket küldhet vissza az aktor számára. Az aktor üzeneteinek és a rendszer válaszainak megadásával a rendszer határait húzzuk meg.

Az aktor jellemzői

- A rendszerrel kommunikáló személy, hardver elem vagy más rendszer.
- Az aktor a rendszeren kívül áll.
- Használati eseteket indítványozhat, illetve fogadhat. Ad és/vagy kap információt. Az aktor passzív is lehet.
- Az aktor az UML osztályszerű eleme (classifier), vagyis nem objektum.
- Az aktor egy szerep. Egy személy a rendszer használatában eljátszik egy szerepet. Jogosultságot fejezhet ki.
- Egy személy egyszerre lehet több aktor, és egy aktor lehet több személy.
- Jele: pálcikaember

Aktor

Ha két aktor ugyanúgy használja a rendszert, akkor a két aktor ugyanaz. Ha valaki kétféleképpen használja a rendszert, akkor az két aktor.

 PÉLDA

A mikrohullámú sütőnek egyetlen aktora van: Használó. A sütőt bárki használhatja, mégpedig egyformán. A sütőt ugyanúgy használja Anita és Dávid is, ők a rendszer szempontjából Használók.

A tanári levelezőben négy aktor van: Használó, Tantárgyhasználó, Tantárgyfelelős és Rendszeradminisztrátor. Elképzelhető, hogy Gergő Tantárgyfelelős és Rendszeradminisztrátor is egyben.

Az aktor azonosítása

A következő gondolatok, szempontok segítik egy rendszer aktorainak meghatározását:

- Ki/mi használja a rendszert (ember, külső erőforrás)?
- Kinek mi az érdeke? Kinek könnyíti meg az életét a születendő szoftver?
- Milyen dolgok történnek?
- Ki fogja a rendszert karbantartani, adatokkal feltölteni?
- Ki mit használhat a rendszerből (jogosultság)?

 CASE

Hozzunk létre a HE nézetben egy HE diagramot, neve main!

- Use Case View / Jobb egér / New diagram / Use Case diagram / main.
  Megjelenik egy üres diagramterület.

Tegyünk a main-re egy Használó aktort!

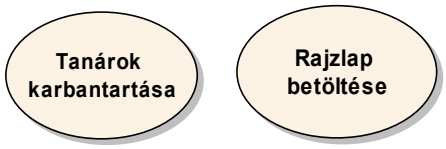
- UML toolbar / Use Case / Actor / Kiválaszt. Diagram területre rátesz. Név = Használó.

5.2. Használati eset

A használati eset a szoftver használatának egy értelmes egysége, az aktor kommunikációja, párbeszéde a szoftverrel. A használati eset a rendszer viselkedését írja le a rendszeren kívülről.

A használati eset jellemzői

- A használati esetek vezérlik a fejlesztést.
- Csak a MIT írja le, a HOGYAN-t nem.
- Ha egy használati esetet elindítottak, akkor az teljes egészében végrehajtódik.
- Mindig az aktor indítványozza.
- A használati eset az UML osztályszerű eleme, melynek egy példánya a forgatókönyv.
- A forgatókönyv a használatnak egy konkrét esete. A forgatókönyvet interakció diagramokkal lehet modellezni, kifejteni.
- A használati eseteket szövegesen le kell írni, dokumentáli kell.
- Jele: ellipszis



Használati esetek

Felhasználói cél – használati eset

Különbséget kell tennünk a felhasználói cél és a használati eset között.

Felhasználói célnak nevezzük a felhaszálónak azt a célját, melyet a szoftver használatával szeretne elérni. A felhasználói célok használati esetekre bontandók: a cél elérése, illetve a feladat megoldása érdekében a felhasználó konkrét használati eseteket hajt végre. A használati esetek a szoftverbe előre beépített lehetőségek, melyeket a felhasználó (aktor) indítványozhat.

 PÉLDA

Felhasználói cél: a felhasználó (rajzoló gyerek) szeretne készíteni egy szép rajzot édesanyja születésnapjára. Ahhoz, hogy ezt a célt elérje, a következő használati eseteket kell a szoftvertől igénybe vennie: Háttérszín beállítása; Szabadkézi rajz kiválasztása; Alakzat rajzolása; Ellipszis kiválasztása; Alakzat rajzolása; (az előzőek variálása) Rajzlap elmentése; Rajzlap nyomtatása.

Forgatókönyv

Egy forgatókönyv (eseményfolyam) a használati eset egy konkrét végrehajtása (példánya). Egy használati esetnek elvileg számtalan forgatókönyve lehetséges.

 PÉLDA

A TanLev-ben a *Bejelentkezés* haszálati eset három forgatókönyve például:

- Gergő beírja felhasználi nevét és jelszavát. Lenyomja az OK gombot. Vége a HE-nek, a rendszerbe való belépés megtörténik. Ez a HE fő folyama.
- Dénes beírja felhasználói nevét. Lenyomja a Mégse gombot (meggondolta magát). Vége a HE-nek, a rendszerbe való belépés nem történik meg. Ez a HE egy alfolyama.
- István beírja felhasználói nevét és jelszavát. Lenyomja az OK gombot. Jön egy üzenet: „Nem jó a jelszó”. István lenyomja a Mégse gombot (rájött, hogy ide nem lehet „csak úgy” belépni). Vége a HE-nek, a rendszerbe való belépés nem történik meg. Ez a HE egy alfolyama.

Természetesen még akárhány forgatókönyvet meg lehetne adni. Ez a három azonban teljességgel körvonalazza a szoftver működését.

5.3. Kapcsolatok

Kapcsolat két aktor között

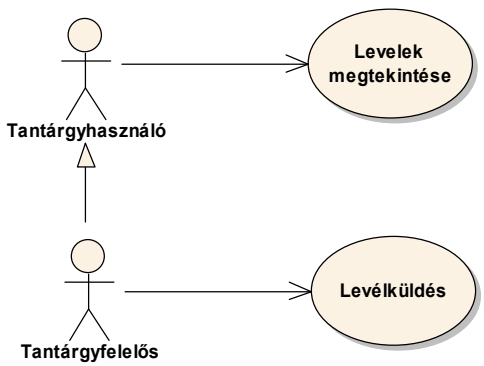
Két aktor között öröklési és kommunikációs (társítási) kapcsolat lehetséges.

Öröklési kapcsolat

A leszármazott aktor minden használati esettel kapcsolatban áll, amellyel az őse kapcsolatban áll.

 PÉLDA

A Tanlev szoftverben a Tantárgyfelelős egyben Tantárgyhasználó is. A tantárgyfelelős megnézheti a leveleket, de ezen kívül levelet is küldhet saját tanárainak:



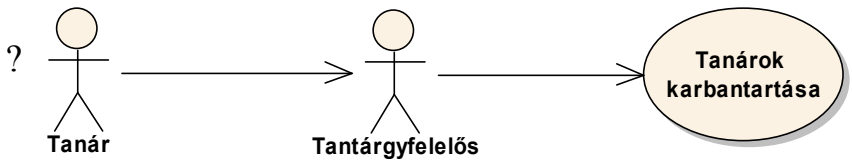
Kommunikációs (társítási) kapcsolat

Két aktor közötti kommunikációs kapcsolat csak információ jellegű, a rendszernek nem eleme.

 PÉLDA

A TanLev szoftverben a Tanár alapvetően nem aktor. Őt a rendszer nyilvántartja, de nem feltétlenül használja a rendszert (csak ha kap jogosultságot). Az a tanár, aki használja a rendszert, Használó. A Használó azonban bárki lehet, nem csak tanár.

De: A Tantárgyfelelős felviszi saját tanárai adatait, amihez adott esetben szükség lehet a Tanár és a Tantágyfelelős kommunikációjára. Ha a rendszerben ezt a kommunikációt fontos kifejezni, akkor a két aktor kapcsolatban álljhat egymással. De fontos megjegyezni, hogy a Tanárnak nincsenek használati esetei, ő csak információs jelleggel van a dokumentációban



Az aktorok közötti kommunikáció általában zavaró, kerüljük használatát!

Kapcsolat aktor és HE között

Egy aktor és egy HE között csak társítási (kommunikációs) kapcsolat lehetséges. Az aktor és a HE között általában párbeszéd van. A nyíl iránya a kezdeményezést mutatja:

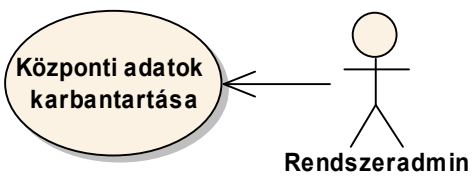
- Kezdeményező aktor esetén a kapcsolat a HE felé irányul;
- Megszólított aktor esetén a kapcsolat a HE felől az aktor felé irányul.
- Résztvevő aktor esetén mindkét aktor kezdeményezhet, ekkor a kapcsolat kétirányú. Ekkor nem teszünk nyilat a vonalra.

Egy aktor akárhány HE-vel kapcsolatban állhat. Egy HE-vel több aktor is kapcsolatban állhat. Ha egy aktor nem áll kapcsolatban egyetlen HE-vel sem, akkor ő nem tartozik a rendszerhez.

 PÉLDA

A TanLev szoftverben a Rendszeradminisztrátor a *Központi adatok karbantartása* HE-ben kezdeményező aktor, hiszen a karbantartást a kiadott információk (adatok) alapján végzi. Például csak akkor vesz fel egy tanárt, ha még nincs felvéve.

Egy folymatosan működő óra kezdeményező aktor, hiszen állandóan „lökdösi” a rendszert, hogy letelt egy adott idő. Ha a stoppert be és le lehet állítani, akkor a stopper egy résztvevő aktor.



Kapcsolat két HE között

Két használati esett között három féle kapcsolat lehetséges: tartalmazás (include), bővítés (extend) és öröklés (generalize). Alapértelmezés: tartalmazási kapcsolat.

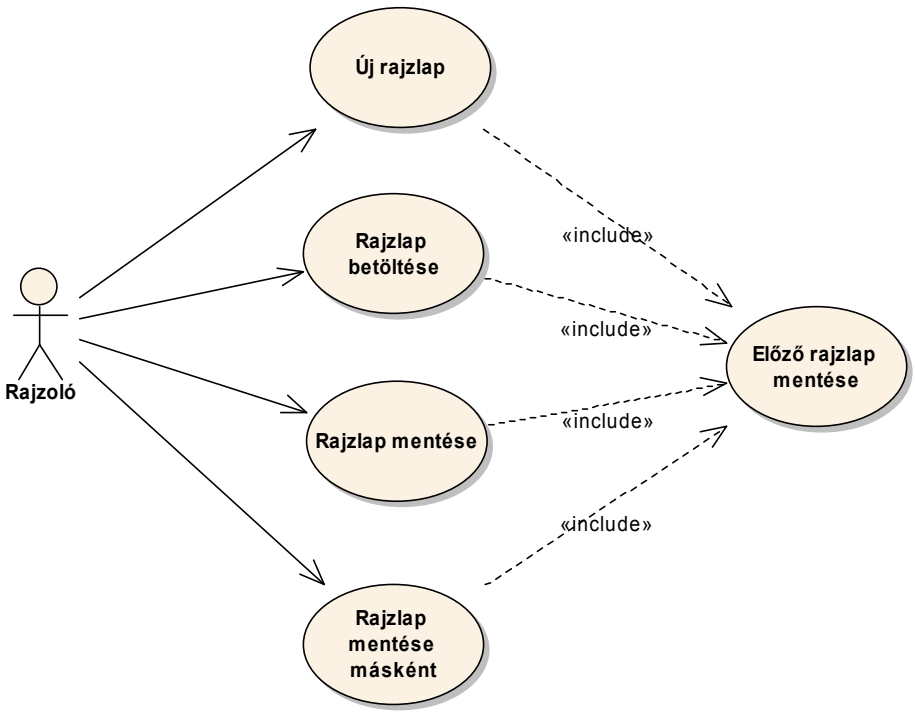
Tartalmazás – include

Két használati eset között a kapcsolat tartalmazási, ha a tartalmazó HE végrehajtásakor a tartalmazott HE feltétel nélkül végrehajtódik. Ezzel elkerülhető a közös viselkedés többszöri meghatározása, a redundancia. Ilyenkor a HE nincs feltétlenül kapcsolatban az aktorral.

A és B két használat eset. Ekkor az A tartalmazza B-t jelentése: A-ból B feltétel nélkül végrehajtásra kerül.


PÉLDA

KissDraw: Az *Új rajzlap* HE minden esetben meghívja az Előző rajzlap mentése HE-t, amely megadja a lehetőséget a Rajzolónak, hogy az előzőleg szerkesztett rajzot elmentse.



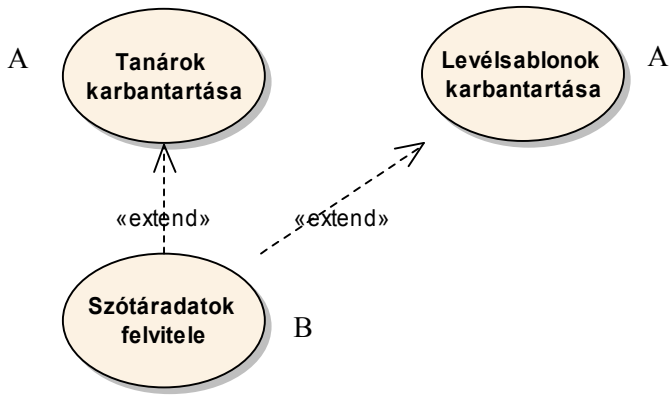
Bővítés – extend

Két használati eset között a kapcsolat bővítési, ha az egyik HE kibővíti, kiegészíti a másikat. Ezáltal a fő használati eset logikáját a bővítő nem zavarja. A bővítő HE végrehajtása feltételes. A kivételes viselkedést külön kezeljük, hogy ne zavarja a normális viselkedést.

A és B két használat eset. Ekkor a „B extends A” (B kibővíti A-t) jelentése: A végrehajtásakor bizonyos feltételek mellett B is végrehajtásra kerül.

PÉLDA

A TanLev szoftverben a *Tanárok karbantartása* vagy a *Levélsablonok karbantartása* adott esetben maga után vonja a *Szótáradatok felvitele* HE-t.



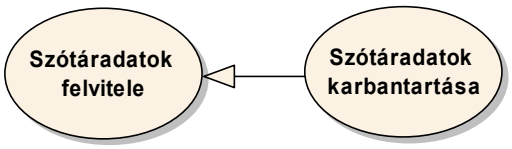
Általánosítás (öröklés) – generalize

Két használati eset között a kapcsolat öröklési, ha az egyik HE örökli a másik HE összes funkcionalitását, és azt felülírja. A B haszálati eset az A használati eset leszármazottja. Speciális viselkedés (B olyan, mint A, de...). Az utód B hozzáad az ős A viselkedéséhez, bizonyos dolgokat felülbírál.

Ilyenkor célszerű az egyes használati eseteket osztályként implementálni: származtassuk az A használati esetet megvalósító osztályból a B használati esetet megvalósító osztályt!

PÉLDA

TanLev: A *Szótáradatok felvitele* HE leszármazottja a *Szótáradatok karbantartása*. A felviteli funkcióhoz még hozzátesszük a törlési és módosítási lehetőségeket.



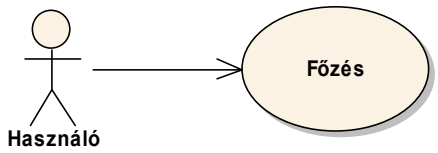
5.4. HE diagram

A használatieset-diagram elemei:

- Aktorok
- Használati esetek
- Lehetséges kapcsolatok:
  - Két aktor között: öröklési és kommunikációs.
  - Aktor és HE között: társítási (kommunikációs) kapcsolat. Van kezdeményező és résztvevő aktor.
  - Két HE között: tartalmazás (include), kiterjesztés (extends), öröklés.

PÉLDA

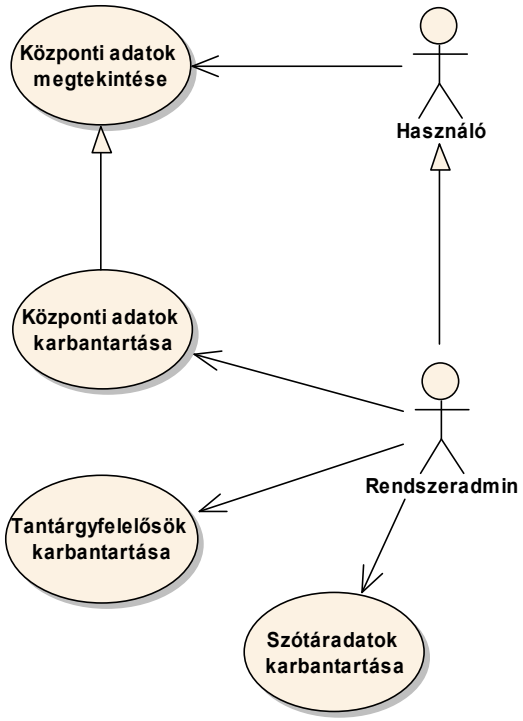
A következő ábra a mikrohullámú sütő (Mikro) használatieset diagramját mutatja. Itt mindössze egyetlen aktor és egyetlen HE van:



A mikrohullámú sütő használatieset diagramja

A következő ábrán a Tanári levelező (TanLev) használatieset diagramjának egy részlete látható, a Rendszeradmin használati eseteit mutatja be. A Rendszeradmin három lényeges dolgot „akar” a rendszertől:

Megtekinti és karbantartja a központi adatokat (konzultációs központokat, tantárgyakat stb.); Karbantartja a tantárgyfelelősöket; és Karbantartja a rendszer szótáradatait.



Tanári levelező használatieset diagramja (részlet)

5.5. HE modell

A használatieset-modell a teljes rendszer viselkedésének a leírása. A HE modell alapján a felhasználó érti, hogyan kell használni a szoftvert. A fejlesztés HE centrikus; a használati esetek a teljes fejlesztés során központi szerepet játszanak. A HE modell elemei: HE diagramok és leírások.

Használati esetek vezérlik a következő dolgokat:

- felhasználói igények (követelmények) meghatározása és ellenőrzése;
- fejlesztés ütemezése;
- analizálás és tervezés;
- tesztelés;
- felhasználói dokumentáció struktúrája;
- átadás.

A HE modell első verzióját a felmérés munkafázisban adjuk meg.

5.6. Mi a jó használati eset?

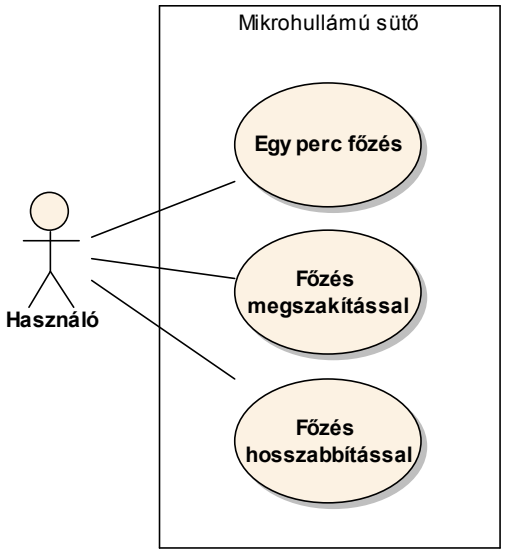
A rendszer használati esetei nem mindig egyértelműek. Sok esetben „csak” a részletességgel van a probléma, de az is elképzelhető, hogy több különböző megközelítés is lehetséges. Ugyanahhoz a feladathoz többféle HE diagram is felrajzolható. Fontos azonban, hogy a használati esetek lefedjék a rendszer funkcióit.

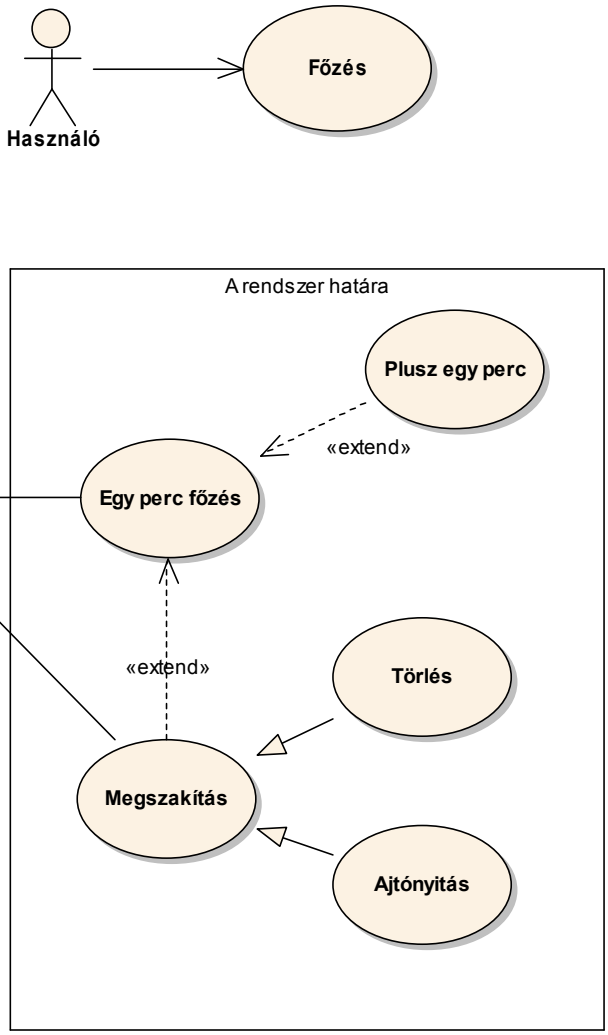
Részletesség

Nézzük a részletességet: Jacobson egy 10 emberéves munkához 20 darab használati esetet definiált. Martin Fowler hasonló esetben kb. 50 használati esetet határozna meg. A Tanárok karbantartása használati esetet például meg lehet adni három HE-ként: Tanárok felvitele, Tanárok törlése, Tanárok módosítása.

Más megközelítés

Elemezzük a következő két HE diagramot! Jól választottuk meg a használati eseteket? Lehet több variáció? Van optimális megoldás?





HE diagram változatok – Melyik a jó?

Egy másik HE "együttes": Indítás, Időmódosítás, Leállítás.

Kérdés: Az ajtónyitás használati eset?

5.7. A használati eset dokumentációja

A HE diagramhoz tartozik egy HE dokumentáció, melyben minden egyes használati esetről meg kell adnunk a HE szöveges leírását, a HE aktorait, valamint szükség szerint a jellemző forgatókönyvek leeírásait és szekvenciadiagramjait. A dokumentációt ajánlatos egy – az adott szoftverfejlesztés keretében használt – sablon (projektszabvány) szerint készíteni.

Részletesebben:

Leírás: Érthető szöveg, melyben röviden felvázoljuk a HE célját, jellemzőit.

Aktor: A HE használója (használói).

Megszorítások (korlátozások): Szabályok, amelyek megtehetők vagy meg kell tenni.

- Előfeltételek, amelyeknek teljesülniük kell a HE végrehajtása előtt.
- Utófeltételek, melyek majd teljesülnek a HE végrehajtása után.
- Alapfeltételek: Olyan dolgok (állítások, feltételek), melyek az egész HE során igazak.

Forgatókönyvek (eseményfolyamok): Egy forgatókönyv a HE egy konkrét végrehajtása. Cél, hogy a HE minél több jellemző ágát bejárjuk. Minden egyes, az előzőtől különböző forgatókönyv megadása közelebb visz a megoldáshoz. A forgatókönyvek segítségével újabb osztályok, szabályok fedezhetők fel.

A használati eset elvileg vezérlőszerkezeteket és kivételeles eseteket is tartalmazhat. A struktúra pontos megadása az implementáció feladata. Itt a cél az, hogy lényegében megértsük a HE működését, közelebb kerüljünk a szoftver logikájához, és feltárjuk az osztályok felelősségeit. Megadhatjuk a **fő végrehajtási folyam**ot, vagyis a használati eset normális lefutását (basic path / happy path). Ezt követően le kell írni az **alfolyamok**at (alternatív folyamokat).

Interakció diagramok: Egy forgatókönyv alátámasztható interakciódiagrammal (szekvencia vagy együttműködési).

A HE végrehajtási struktúráját (a vezérlőszerkezeteket) aktivitásdiagrammal tudjuk pontosan megadni, ha szükséges.

A rendszerben mindössze egyetlen aktor, és egyetlen használati eset van.

 PÉLDA

Nézzük a Mikrohullámú sütő *Főzés* használati esetét!

Főzés – használati eset

Leírás

A Használó beteszi az ételt a mikróba, és azt egy adott ideig főzi. A főzés használati eset összetett, annak számtalan forgatókönyve lehetséges.

Előfeltétel

Nincs.

Utófeltétel

Az étel meg van főzve.

Egy perc főzés – forgatókönyv

A Használó kinyitja az ajtót, beteszi az ételt, becsukja az ajtót. Megnyomja a Perc plusz gombot. Elindul a főzés, a lámpa ég. Egy perc lejártával befejeződik a főzés, a lámpa kialszik. Használó kinyitja az ajtót, a lámpa kigyullad. Kiveszi az ételt. Becsukja az ajtót, a lámpa elalszik.

A forgatókönyvhöz tartozó szekvenciadiagramot lásd az *Interakciódiagramok* fejezetben.

Három perc főzés, közben 1:33-nál ajtónyitás – forgatókönyv

A Használó háromszor lenyomja a Perc plusz gombot. Az elsőre elindul a melegítés. 1:33-nál kinyitja az ajtót, mire a főzés félbeszakad. Később becsukja az ajtót, a főzés újra indul onnan, ahol abbamaradt. A három perc lejártával a sütő leáll, a lámpa kialszik. A használó kinyitja az ajtót.

Két perc főzés, közben 1:50-nél törlés – forgatókönyv

A Használó beteszi az ételt, becsukja az ajtót. Egyszer lenyomja a Perc plusz gombot. Erre elindul a főzés. Fél perc lejártával megnyomja a Törlés gombot. Erre befejeződik a melegítés.

5.8. A használati eset megvalósítása

A használati eset implementálásához általában a HE-nek megfeleltetünk egy kontroll osztályt, vagy egy kontroll osztályban egy metódust. A kontroll osztály sok esetben egybeesik a felhasználói felület egy elemével (ablakával) illetve egy gomb lekezelő metódusával. Az aktor közvetlenül az ablakkal kommunikál, az ablak pedig a kontroll objektumnak továbbítja a kéréseket. A kontroll objektum a felelősségeket leosztja más osztályokra.

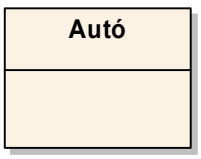
6. Osztálydiagram

Osztálydiagram (class diagram): Olyan diagram, amely az osztályokat és a közöttük lévő társítási és öröklési kapcsolatokat ábrázolja. Az objektumdiagram az osztálydiagram előfordulása, példánya. Az osztálydiagram rögzíti az objektumok közötti kapcsolatok szabályait.

Két osztály közötti társítási kapcsolat főbb jellemzői: ismeretségi vagy tartalmazási kapcsolat (ha tartalmazási kapcsolat, akkor erős vagy gyenge); név; multiplicitás (egy–egy, egy–sok vagy sok–sok jellegű; kötelező vagy opcionális); szerepnév; megszorítás.

6.1. Diagramelemek

Osztály (class)



Az osztály egy objektum minta. Vannak tulajdonságai és viselkedése (metódusai). A rendszer modellezésének végső célja a rendszerben szereplő osztályok és azok kapcsolatainak meghatározása, az osztályok forráskódhoz rendelése és kódolása, a szoftver komponensek konkrét hardver eszközökhöz való hozzárendelése. A modellezés minden változata e cél szolgálatában áll.

Az aktor egy speciális osztály. Az aktor a rendszeren kívül levő szereplő. Az aktorhoz nem tartozik a rendszerben közvetlenül lekódolt osztály, mint szoftvert komponens. Az aktor az osztály egy sztereotípusa.

Interfész (interface)

Az interfész metódusfejeket definiál. Az interfészt az osztály valósítja meg.

Együttműködés (collaboration)

Az együttműködés a használati eset megvalósítása.

Aktív osztály (active class)

Párhuzamos folyamatok esetén az az osztály, amelyik éppen birtokolja az aktív szálat. Jelölése: vastag szélű osztály.

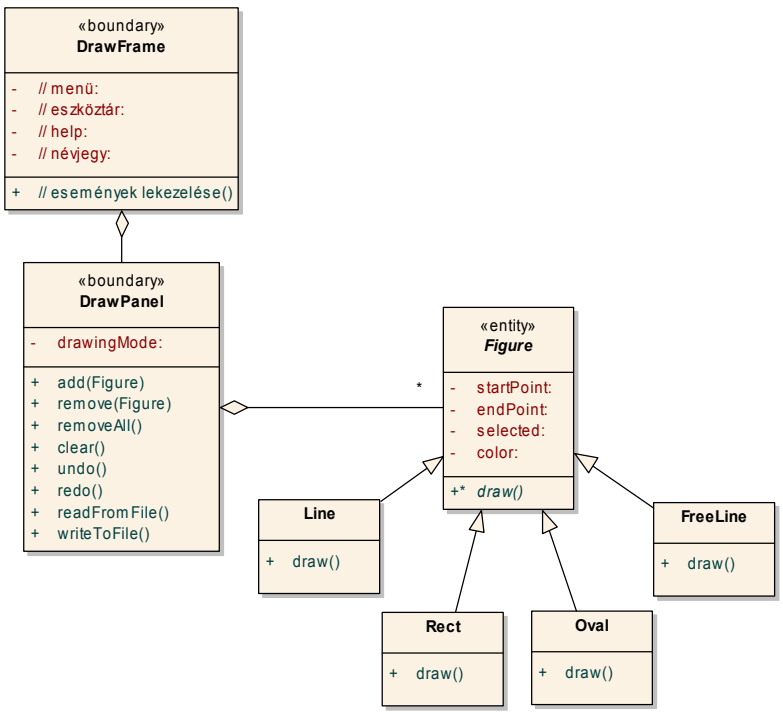
Kapcsolatok

Két osztály társítási kapcsolatát a következők jellemzik:

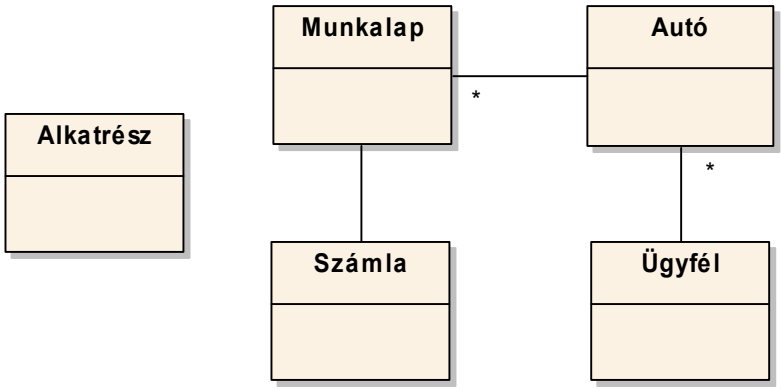
- **Kapcsolat**: Vonalat húzunk a két osztály közé. Az ismeretségi, illetve tartalmazási (erős vagy gyenge) kapcsolatokat ugyanúgy jelöljük, mint objektumok esetében. Opcionális, vagyis nem kötelező megadni.
- **Navigálási irány**: A vonal végein levő nyilak mutatják. Opcionális.
- **Kapcsolat neve és iránya**: A vonalra tesszük nagy kezdőbetűvel, vastag betűsen, aláhúzás nélkül. Opcionális.
- **Multiplicitás (kardinalitás)**: A vonal két végére egy-egy számot vagy számhalmazt írunk: az osztályhoz közel eső szám azt jelenti, hogy a szemközti osztály egy objektumához hány objektum tartozhat ebben az osztályban. Számhalmazt számok és számintervallumok felsorolásával adhatunk meg. A * jelentése: akárhány. Ha nem írunk a vonalra semmit, az 1-et jelent. Például:
  - 3 : pontosan három
  - 0..1 : nulla vagy egy
  - 10..20 : 10 és 20 közé eső szám
  - * : 0..∞, vagyis akárhány
  - 1..* : 1..∞, vagyis legalább egy
  - 1,3,10..* : 1 vagy 3, vagy 10-től kezdve akárhány.
- **Szerepnév**: Két osztály kapcsolata nagyon jól jellemezhető azzal, hogy mi az osztályok szerepe ebben a kapcsolatban. A szerepnevet kisbetűvel írjuk. Opcionális.
- **Megszorítások (kitételek)**: Ilyen például a kapcsolat rendezett volta (ha egy objektummal kapcsolatban álló objektumok valamilyen szempont szerint rendezve vannak). A megszorításokat kapcsos zárójelbe tesszük, például: {rendezett}. Opcionális.


PÉLDA

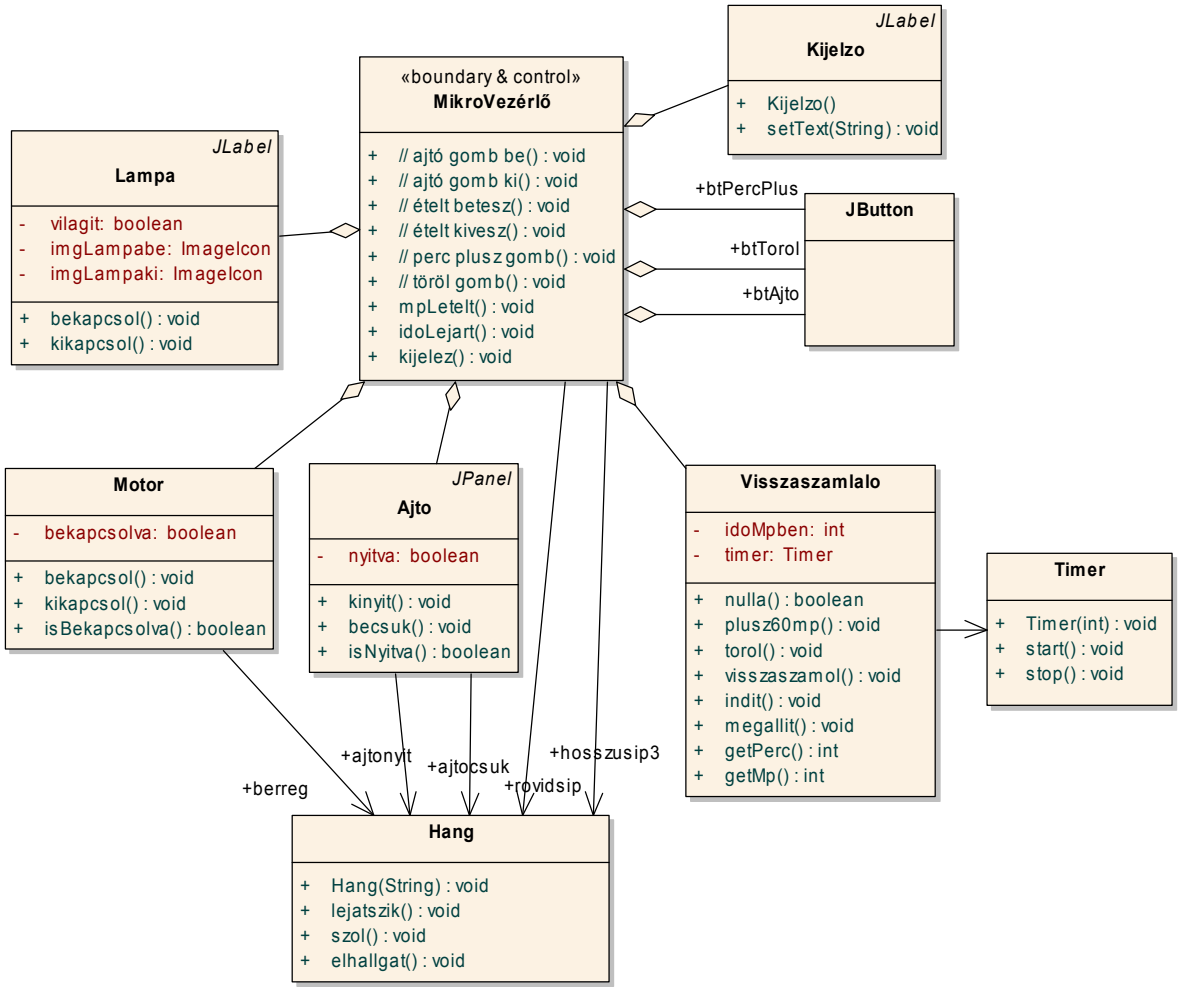
A KisDraw, a Szerviz és a Mikro alkalmazások osztálydiagramját a következő ábrák mutatják.



A KissDraw alkalmazás szakterület osztálydiagramja



A Szerviz alkalmazás szakterület osztálydiagramja



A Mikro alkalmazás osztálydiagramja

7. Interakciódiagramok

7.1. Kétféle interakciódiagram

Az interakciódiagram objektumokat és az azok közötti üzeneteket ábrázolja. Segítségével az objektumok együttműködését, a végrehajtandó üzenetsorokat szemléltethetjük. Egy interakciódiagram tipikusan egy használati eset egy forgatókönyvét írja le. A diagram használatával az objektumok/osztályok közötti felelősségek könnyebben feltárhatók, illetve szétoszthatók. Kétféle interakciódiagram létezik: **szekvenciadiagram** és **együttműködési diagram**. A két diagram ugyanazt az információt tárolja, a különbség megjelenésükben van: míg a szekvenciadiagramon az idő a fontos, az együttműködési diagramon az objektumok kapcsolata az elsődleges információ. A két diagram átalakítható egymásba.

Egy használati eset forgatókönyveit interakciódiagrammal szemléltetjük. A szoftver tervezésekor minden egyes aktor–HE párhoz nulla, egy vagy több interakciódiagramot szokás készíteni, a HE bonyolultságától függően. Az objektumoknak küldött üzenet a megfelelő osztályban megjelenik.

Az interakciódiagram elsődlegesen a használati esetek analizálásának eszköze. Nem a lépéssorozat struktúrája (elágazások, iterációk) a lényeg, hanem az, hogy mely objektumoknak milyen üzenetet küldünk. A küldött üzenet ugyanis az objektum osztályában szerepelni fog. A küldés vonalán kirajzolódnak az osztályok, az osztályok közötti kapcsolatok, valamint az osztályok metódusai. Az interakciódiagramok készítésekor „kipotyognak”az osztálydiagram jellemzői.

Az interakciódiagram alapján az osztálydiagram egy része vagy egésze felépíthető. Mindkét interakciódiagramnak (szelvencia és együttműködési) megvan az előnye. Hogy a fejlesztő melyiket használja éppen, az egyrészt a HE jellegéből fakadhat, másrészt ízlés dolga.

7.2. Szekvenciadiagram

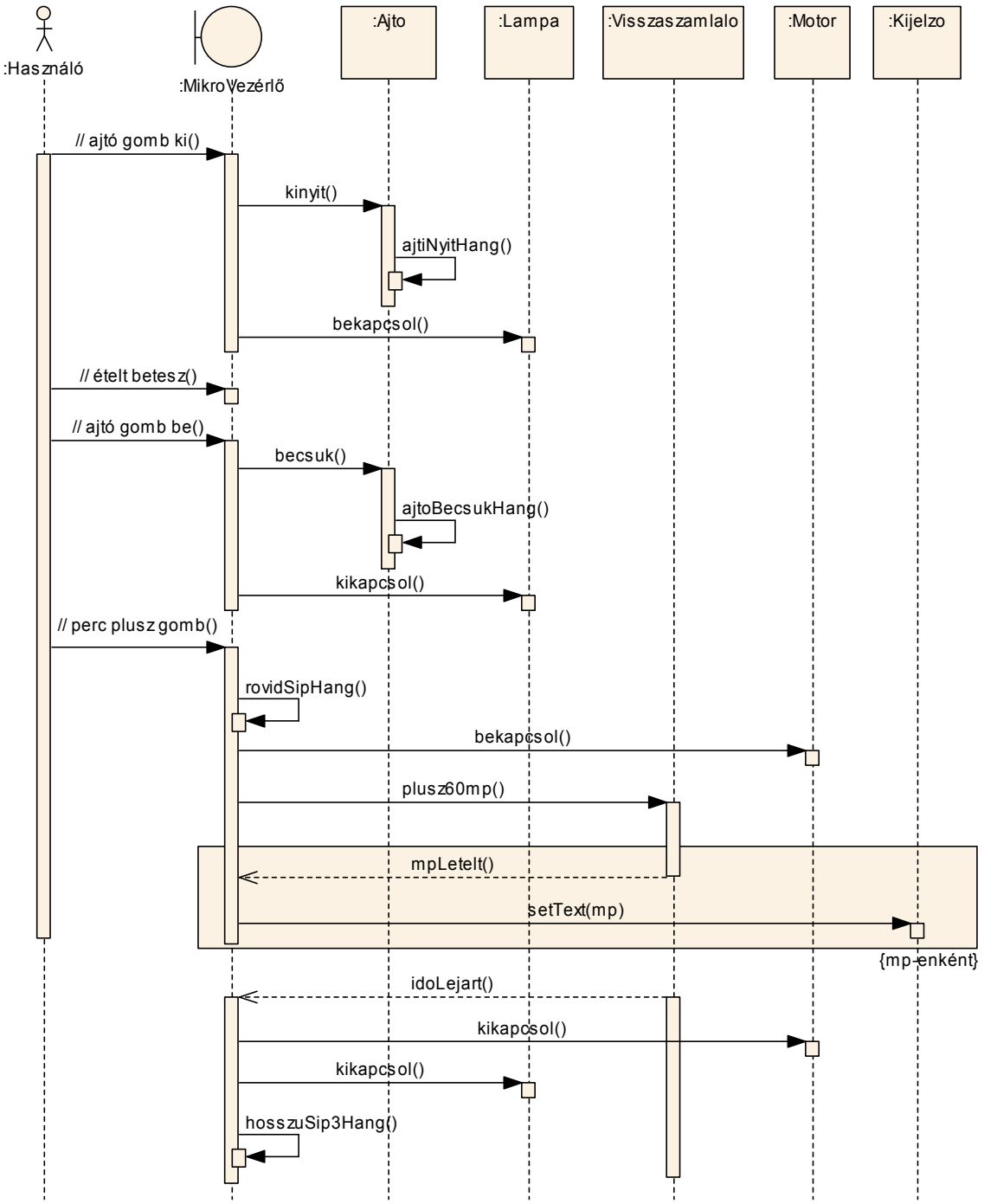
Szekvenciadiagram segítségével leginkább szekvenciális lépések sorozatát ábrázolhatjuk. A szekvenciadiagram nem alkalmas bonyolultabb vezérlőszerkezetek, struktúrák ábrázolására, inkább egy előre rögzített forgatókönyvet szemléltetünk vele.


PÉLDA

Az ábrán a mikrohullámú sütő *Főzés* használati esetének *Egy perc főzés* forgatókönyvének szekvenciadiagramját látjuk. A használó egy percig főz. Indulásképpen a sütő nem működik (a motor ki van kapcsolva), az ajtó csukva van.

- A Használó megnyomja az ajtó gombot. Erre a MikroVezérlő utasítja az ajtót: ajto.kinyit(). Az ajtóba az van beprogramozva, hogy ha őt kinyitják, akkor szólaltasson meg egy hangot. A hang megszólaltatásának kivitelezését későbbre halasztjuk – ezt az ajtó egyelőre elintézi egy önmagának szóló üzenettel: ajtoNyitHang(). Ezután a MikroVezérlő bekapcsolja a lámpát, és egyelőre nyugalmi állapotba kerül.

- A Használó beteszi az ételt, majd megint nyugalom következik. Az étel betételét egyébként nem fogjuk beprogramozni, ennek a felhasználói akciónak csak szimbolikus értéke van.
- Most a Használó megnyomja az ajtócsukó gombot, mire a MikroVezérlő becsukja az ajtót, hangot ad és kikapcsolja a lámpát.
- Ezután a Használó megnyomja a Perc plusz gombot. Erre megszólal egy rövid síp (valószínűleg a sípolást egy Hang objektum fogja végezni, de ezt még nem tudjuk pontosan). A mikroVezérlő bekapcsolja a motort, és egy percre állítja a Visszaszámláló objektumot (a visszaszámlálás az ő dolga lesz). A visszaszámláló másodpercenként visszajelez a MikroVezérlőnek, hogy lejárt(). A MikroVezérlő válaszképpen kijelzi az időt. Amikor végleg lejárt az idő, akkor a MikroVezérlő kikapcsolja a motort, a lámpát, és három hosszú síphangot ad.



Szekvenciadiagram – Egy perc főzés HE

A szekvenciadiagramon az idő a fontos. Az X tengelyen vannak az objektumok, az Y tengely az életvonal (object lifeline). Az idő fentről lefelé telik. Két objektum között itt nem mutatható ki közvetlenül a kapcsolat, a kapcsolat az üzenetek által érzékelhető (üzenet csak kapcsolat mentén lehetséges). Az üzenetek sorszámozhatók.

A szekvenciadiagramon bal oldalon szokás elhelyezni az aktort, és amennyiben lehetséges, az üzeneteket balról jobbra szokás küldeni.

A diagram elemei

Objektum

Az objektumok az X tengely mentén helyezkednek el. Sztereotípussal jelölhetjük, hogy az objektum aktor, határ, entitás vagy egyéb. Lehetőség szerint az aktorok és a kliensek balra, a szerverek (az üzenetek fogadó objektumai) jobbra helyezkedjenek el!

Életvonal

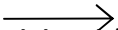
Az objektum életvonala egy függőleges, szaggatott vonal az objektumtól lefelé. Mutatja, hogy az objektum él. A legtöbb objektum a forgatókönyv teljes időtartama alatt él, ekkor az életvonal fentről lefelé nem szakad meg. Ha egy objektum a forgatókönyv ideje alatt születik, akkor azt a *create* vagy *new* sztereotípusú üzenettel jelezhetjük, és az objektumot csak az üzenet végén tesszük ki. Az objektumot a *destroy* sztereotípusú üzenettel szüntetjük meg.

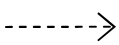
A vezérlés fókusza

Egy objektum a vezérlés fókuszában van, ha az ő metódusa fut, vagy éppen visszavár egy metódushívást. A vezérlés birtoklását egy függőleges, kerkeny téglalap jelzi az objektum életvonalán. Ez alatt az idő alatt az objektumnál van a vezérlés, az objektum éppen működik, aktív.

Üzenetek

Az objektumok közötti üzeneteket vízszintes nyíllal jelöljük, rajta az üzenet neve.

- **Szinkron üzenet:** Beágyazott vezérlés, tipikusan egy metódushívás. Az üzenetet lekezelő metódus lefut, mielőtt a hívóhoz visszakerül a vezérlés. Jele tömör fejű nyilacska:
- **Aszinkron üzenet:** A hívó által küldött aszinkron üzenet nem tér vissza a hívóhoz. A hívó az üzenet elküldése után folytatja működését. Általában a válasz (eredmény) szintén aszinkron üzenetként jön vissza. Jele nyílt hegyű nyilacska: 
- **Visszatérés:** Minden üzenetnek lehet egy visszatérés párja. Egy metódus visszatérését nem kötelező jelölni a diagramon. Alapértelmezésben a szinkron üzenet visszatér a hívás helyére, még a következő hívás előtt. A metódusvisszatérés jele szaggatott, nyílt hegyű nyilacska:



Elágazás

Az üzenet neve előtt megadható egy feltétel (guard). A feltételt szögletes zárójelbe tesszük. Ebben az esetben az üzenet csak a feltétel teljesülése esetén hajtódik végre. Például:

```
[nyitva] becsuk()
```

Iteráció

Az üzenet neve előtt megadható egy csillag és egy feltétel. A feltételt szögletes zárójelbe tesszük. Ebben az esetben az üzenet annyiszor hajtódik végre, ahányszor a feltétel teljesül. Például:

```
*[idoMp-ben != 0] setText()
```

A szekvenciadiagramhoz szöveges magyarázatot lehet elhelyezni a diagram bal oldalán. Itt megjegyzéseket (pl. két üzenet közt eltel időt) és megszorításokat (pl. if motor.isBekapcsolva) lehet megadni.

A szekvenciadiagram építése

Az ábra a mikrohullámú sütő *Főzés* használati esetének *Egy perc főzés* forgatókönyvét szemlélteti. Ahogy a forgatókönyvet összeállítjuk, folyamatosan „felbukkannak" a megfelelő osztályok, az osztályok közötti kapcsolatok, és az osztályok metódusai. Az egyes objektumoknak nem kell nevet (azonosítót) adni, elegendő az osztályuk feltüntetése.

A diagram bal oldalán szerepel a Használó. A Használó mindig egy határ objektummal áll kapcsolatban. A mikrovezérlő objektumot határ objektumnak tüntettük fel, bár legalább annyira vezérlő objektum. A vezérlő ablakot nem bontjuk fel részekre, mert az meglehetősen rontaná az áttekinthetőséget.

A HE előfeltétele, hogy a sütő nem működik, és csukva van az ajtó. Ez annyiban számít, hogy az egyes üzenetküldéseknél nem kell a sütő állapotát vizsgálni, vagyis nem tüntetünk fel szelekciókat. Ez nem baj, mert a szekvenciadiagram segítségével elsősorban azt akarjuk feltérképezni, hogy „ki üzen kinek". Az egyes metódusok pontos struktúráját főleg az állapot- és aktivitásdiagramokkal fogjuk megállapítani.

 CASE

Alakítsuk ki a Mikro program *Egy perc főzés* használati esetének szekvenciadiagramját!

A Használó megnyomja az ajtó gombját. A gomb lenyomását a MikroVezérlő gomblenyomó metódusa kezeli le – a metódus nevét most fonetikusan írjuk, és megjegyzésbe tesszük. Jó gyakorlat, hogy azon metódusok neveit, amelyeket még nem döntöttünk el pontosan, megjegyzésbe teszük.

Vegyünk fel tehát egy MikroVezérlő osztályt, benne egy // ajtó kinyit metódust! Vonszoljuk rá a MikroVezérlő osztály egy példányát a szekvenciadiagramra! Kössük össze a Használót és a MikroVezérlőt egy üzenettel (Message), az üzenet nevénél a rendszer felkínálja az objektum osztályának metódusait. Válasszuk ki az előbb betett metódust!

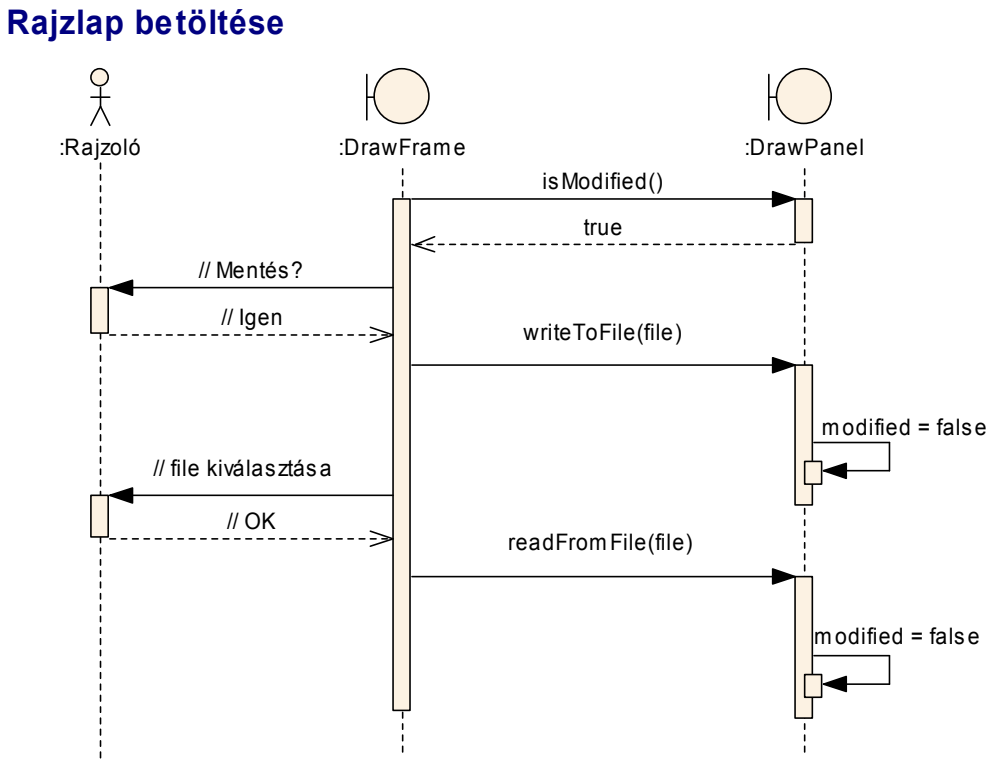
Az ajtónyitó gomb megnyomására a vezérlő kinyitja az ajtót. Ehhez kell egy ajtó, és kell egy kinyit metódus. Vegyünk fel tehát egy Ajtó osztályt a készülő osztálydiagramon! Az Ajtó osztály egy példányát tegyük rá a szekvenciadiagramra! Tegyünk most egy üzenetet a MikroVezérlő és az Ajtó közé! Az üzenet nevének kitöltésekor nyomjuk le a New gombot (new message), és vegyünk fel az Ajtó osztályban a kinyit metódust! Válasszuk a most beírt metódust az üzenet nevének!

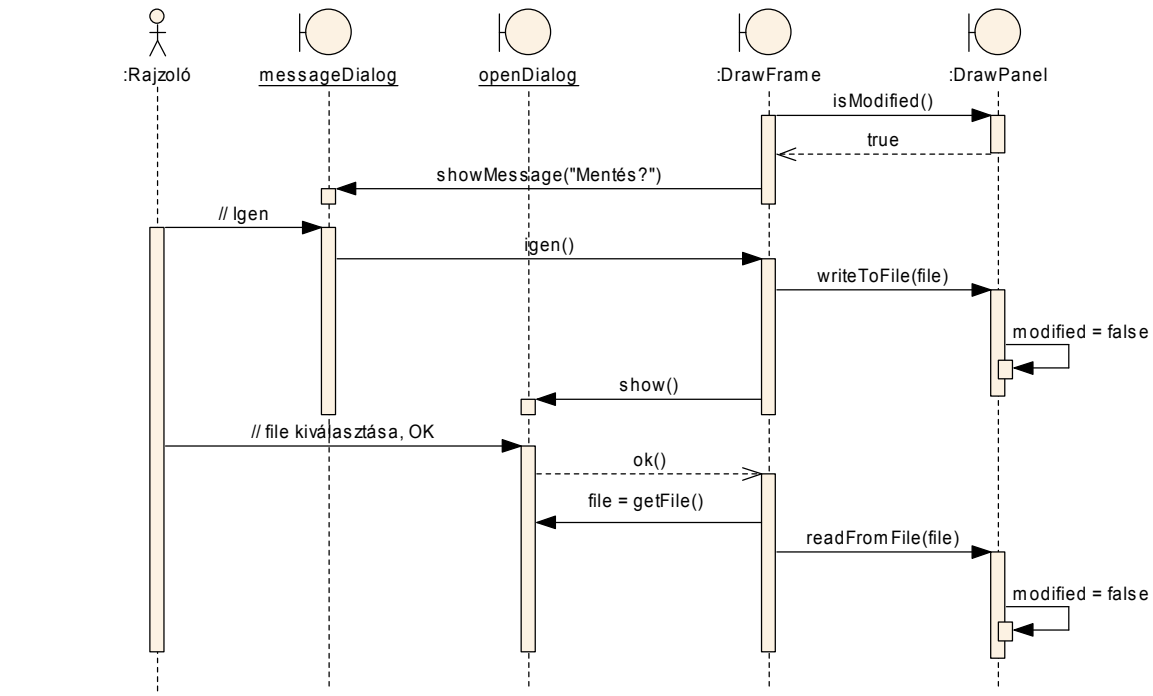
Minden egyes lépésnél tehát át kell gondolnunk, milyen osztályra, annak milyen metódusára van szükségünk.

Több szekvenciadiagram több dologra világít rá. Végül felépül az osztálydiagram.

PÉLDA

A következő ábra a KissDraw program *Rajzlap betöltése* használati esetének egy forgatókönyvét szemlélteti. A szekvenciadiagram a fejlesztés korai fázisában készült, a FileManager osztály még nem szerepel rajta. A DrawFrame a Rajzolóval közvetlen módon kommunikál. Egy implementációközelibb változatban szerepeltethetnénk a különböző dialógusokat is. Ezeket a DrawFrame megjelenítene, atán eltüntetne. Kérdés azonban, hogy ez a részleteség nem hátráltat-e a probléma megértésében?



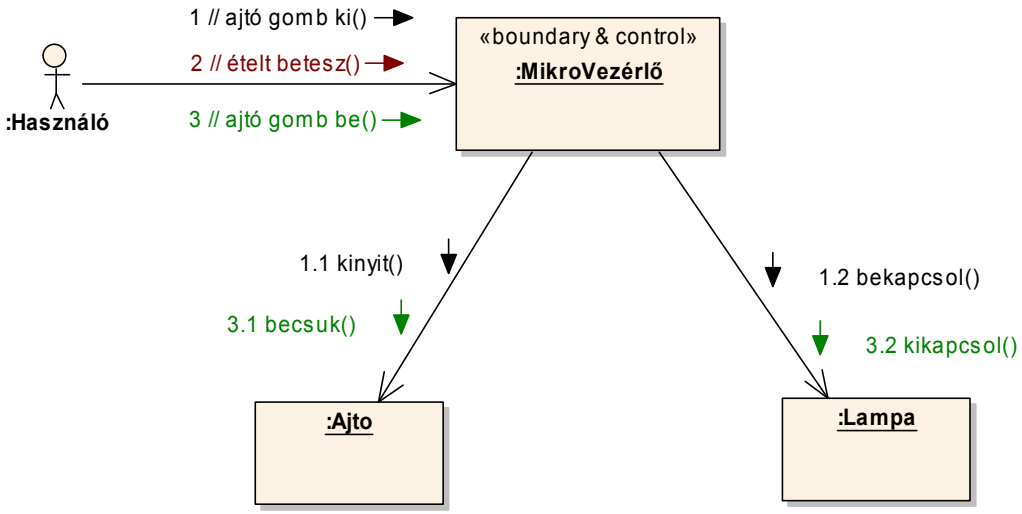


7.3. Együttműködési diagram

Az **együttműködési diagram** (collaboration diagram) egy gráf, ahol a csúcsok az objektumok, az élek a kapcsolatok. Itt nem az idő, hanem az objektumok közötti kapcsolatok a fontosak. Az üzenetek többszintesen sorszámozhatók (ezek a szekvencia számok), ily módon az idő itt is kifejezhető...


PÉLDA

A következő ábra a Mikro Egy perc főzés forgatókönyvének elejét ábrázolja.



Megjegyzés:
Az együttműködési diagram az EA-ban eléggé nehézkes. Nem lehet két üzenet közé beszúrni egy másikat. Nem lehet váltani a két interakciódiagram között.

8. Állapot-átmeneti diagram

8.1. Az állapot-átmeneti diagram szerepe a modellezésben

Az **állapot-átmeneti diagram** (state transition diagram) egyetlen osztály (annak egy előfordulásának) dinamikus viselkedését, a külvilággal való kapcsolatát ábrázolja. Az állapot-átmeneti diagram egy gráf, melynek csomópontjai állapotok, élei pedig átmenetek. Megadja, hogy az objektum mely események hatására milyen állapotból milyen állapotba kerül.

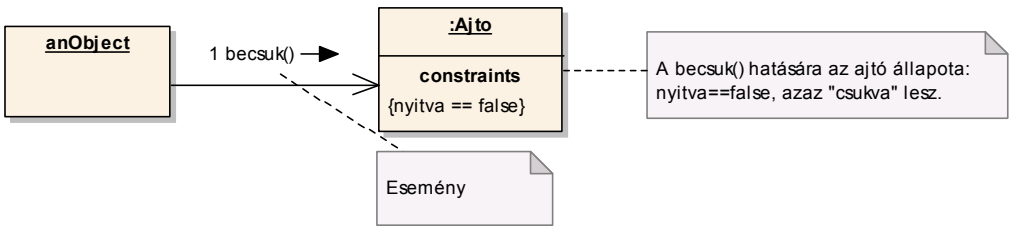
Ha egy objektumnak intenzív a dinamikus működése, akkor az objektum viselkedésének feltárásában az állapotátmeneti diagram igen hatékony eszköz.

8.2. Állapot

A rendszerben minden objektum egy önálló egység, amely a rendszer többi részével kommunikál. Az objektum

- detektálja az eseményeket, illetve
- reagál az eseményekre.

Egy esemény hatására az objektum állapota megváltozhat. Ha például az ajtónak azt üzeni egy másik objekum, hogy becsuk(), akkor az ajtó nyitott állapotból (nyitva==true) csukott állapotba (nyitva== false) kerül:



Egy adott állapotban levő objektum ugyanarra az eseményre mindig ugyanúgy reagál (ugyanazt az akciósorozatot hajtja végre). Az állapot az objektum életének egy szakaszát írja le.

Az állapot (state) lehet:

- Az objektum bizonyos értékeinek és kapcsolatainak aktuális halmaza;
- Időintervallum, amely alatt az objektum vár egy esemény bekövetkezésére;
- Időintervallum, amely alatt az objektum folyamatosan végrehajt egy akciót.

Egy objektumnak különböző jellegzetes állapotai lehetnek, melyeknek nevet adhatunk.

Az állapot jele az UML-ben egy lekerekített téglalap, melynek részei (bármelyik rész elhagyható):

- Név rész: a tégalap felső része, az állapot neve

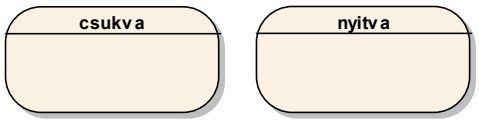
- Belső átmenetek: Olyan akciókat, illetve aktivitásokat tartalmazhat, melyek nem okoznak állapotváltozást (lásd később).

 PÉLDA

A Mikro objektumainak állapotai:

- Ajtó: nyitva / csukva
- Motor: bekapcsolva / kikapcsolva
- Visszaszámláló: áll, ketyeg, függő

Az ajtó állapotainak UML jelölései:



8.3. Állapot-átmenet

Állapot-átmenetnek nevezzük azt a folyamatot, melyben az objektum egy adott állapotából egy egyértelműen megkülönböztethető másik állapotba kerül. Az átmenet lehetséges tulajdonságai:

- Kiváltó esemény (trigger vagy event). Az átmenet a kiváltó esemény hatására következik be. Az esemény egyaránt jöhet kívülről vagy belülről.
- Őrfeltétel (guard). Az őrfeltétel egy logikai kifejezés, amely hivatkozhat az objektum adataira. Az átmenet csak akkor következik be, ha az őrfeltétel igaz.
- Akció (action). Átmenetkor végrehajtódik.

E tulajdonságokat az átmenet vonalán tüntetjük fel a következőképpen (bármelyik elhagyható):

```
trigger [őrfeltétel] / akció
```

Az állapotátmenetet követő akció nem szakítható meg.

Szabályok:

- Egy objektum egyszerre csak egy kiváltó eseményt kezel le;
- Az események sorban állnak;
- A le nem kezelt esemény az objektum számára elvész;
- Az eseménynek időpontja van, nem időtartama.

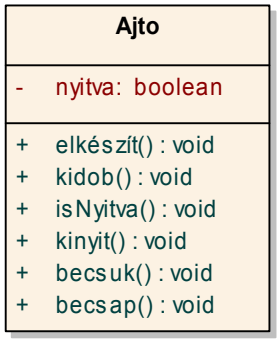
Kezdeti állapot, végállapot

Az objektumnak van

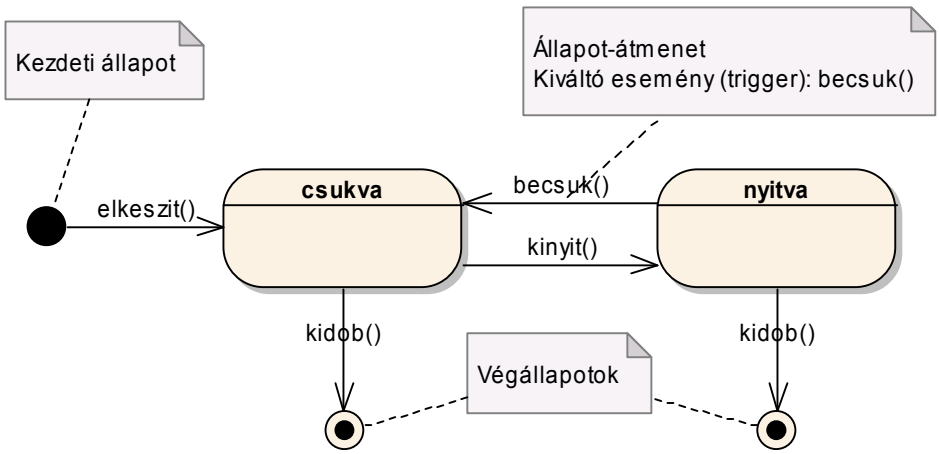
- pontosan egy kezdeti állapota (initial state), melybe az objektum születésekor kerül. Jele egy tömör kör.
- valahány végállapota (final state), ahonnan további állapotváltozás nem lehetséges. Jele egy ökörszem.

 PÉLDA

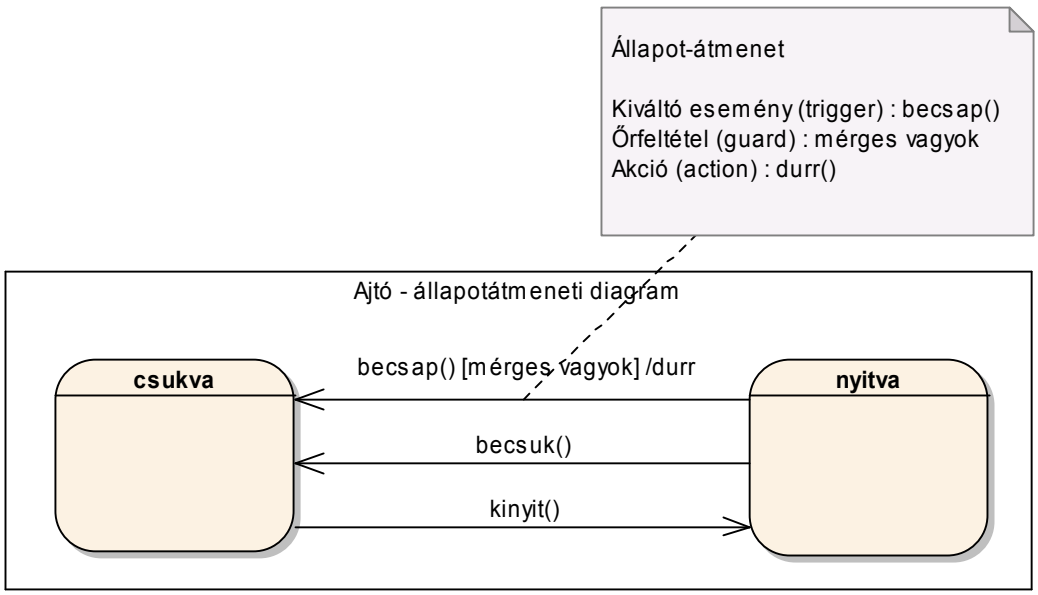
Vegyünk egy képzeletbeli ajtót, melynek osztálydiagramja a következő:



Az Ajtó osztályból hozzunk létre egy példányt. Életét egy állapot-átmeneti diagrammal szemléltetjük. Születéskor –, amikor elkészítik – csukva van. Aztán egész életében nyitják-csukják. Az ajtót ki lehet dobni akár nyitott, akár csukott állapotban.



A következő ábra az ajtó olyan állapot-átmeneteit mutatja, melyben a nyitás és a csukás csendben történik, ellenben a becsapást egy nagy durr kíséri.



A példában a kiváltó események a becsap, becsuk és a kinyit, ezek mind az Ajtó osztály saját metódusai. Az őrfeltétel a [nyitva] és [csukva], a durr az akció.

Események fajtái

Eseménynek vagy kölcsönhatásnak nevezünk bármit, ami hatással lehet egy objektumra. Egy esemény lehet hívási, változási, jel és idő esemény. Az állapot-átmenetet események okozhatják. Egy esemény nem feltétlenül vált ki állapot-átmenetet.

Hívási esemény (call event)

Szinkronizált kérelem (üzenet). Egy másik, kliens objektumtól érkezik. A kliens objektum vár a válaszra.

```
üzenetNeve(paraméterek)
```

Változási esemény (change event)

Egy logikai kifejezés értékének megváltozása.

```
when(kifejezés)
```

Jel esemény (signal event)

Aszinkron üzenet (bármikor jöhet). Két féle lehet:

- rendszerbe kívülről jövő esemény (pl. egér, bill)
- másik objektum jelzése válaszra várás nélkül
  `jelNeve(paraméterek)`

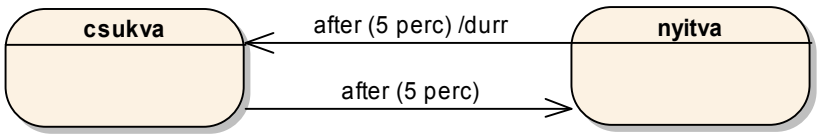
Idő esemény (time event)

Egy konkrét idő elérkezik vagy letelik.

```
after(idő)
```

 PÉLDA

Az előző példában a becsuk egy hívási esemény. Ha az ajtót 5 percenként szeretnénk nyitni-csukni, akkor az állapotátmenet kiváltó eseménye egy idő esemény (after...) lenne:



8.4. Tevékenységek az adott állapotban

Vannak események, melyek az objektumon nem okoznak közvetlen állapotváltozást. Ezeket az eseményeket az állapot doboz akció részébe írjuk a következő formában:

```
Akciócímke / Akció-kifejezés
```

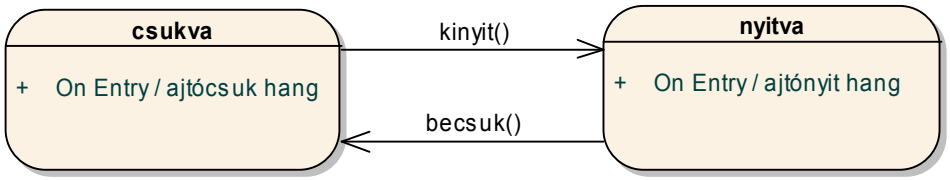
Akciócímkék:

- **On Entry**: az akció az **állapotba való belépéskor** kerül végrehajtásra;
  `On Entry / Akció-kifejezés`
- **On Exit**: az akció az **állapotból való kilépéskor** kerül végrehajtásra;
  `On Exit / Akció-kifejezés`
- **Do Action**: tevékenység, mely az adott állapot fennállása alatt **folyamatosan** fut;
  `Do Action/ Akció-kifejezés`

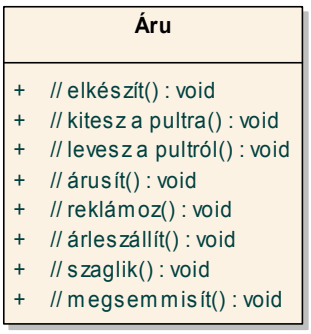
- **[On Event]**: egy adott **esemény hatására** kerül végrehajtásra.
  Itt magát az eseményt írjuk a címke helyére.
  `Esemény (paraméterlista) [feltétel] / Akció-kifejezés`

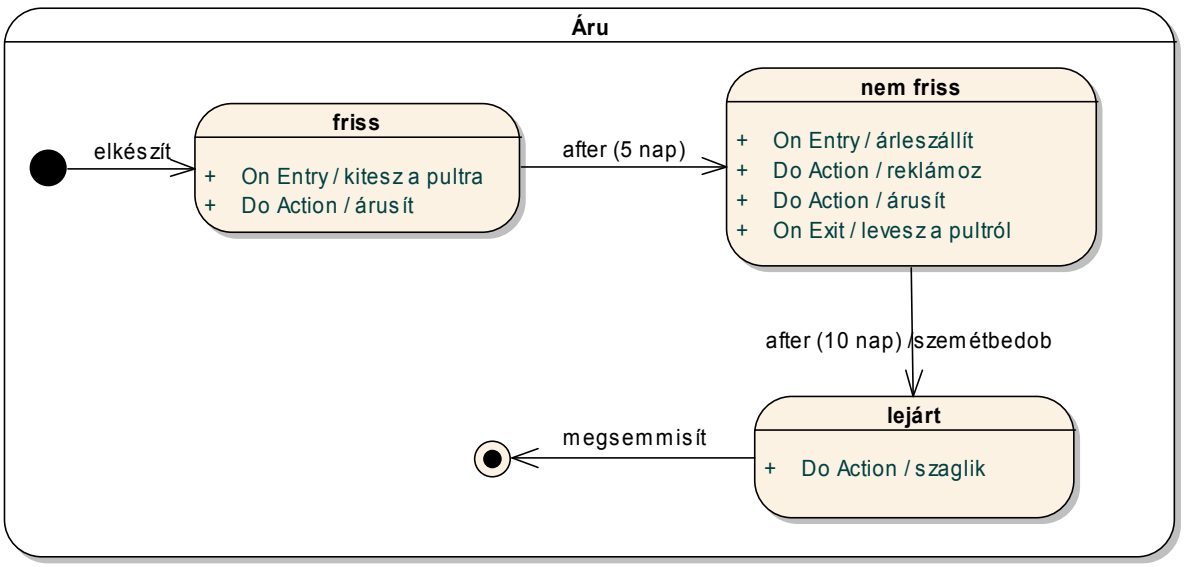
PÉLDA

A következő ajtó esetében a nyitásnak és csukásnak mindig van hangja függetlenül attól, hogyan érkezett ebbe az állapotba:

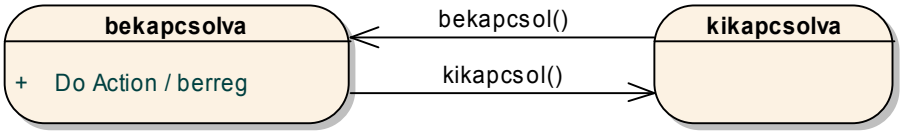


Vegyünk egy Áru osztályt. Az Áru objektumnak három állapota van: friss, nem friss és lejárt. Az állapotok átmeneteit és az állapotváltozásokat kísérő jelenségeket a következő állapot-átmeneti ábráról olvashatjuk le:





A mikrohullámú sütő majd minden osztályának triviális az állapot-átmeneti diagramja. A Motor és a Lámpa például bekapcsolt vagy kikapcsolt állapotban lehet. Az állapotváltozásokat az osztály bekapcsol, illetve kikapcsol metódusainak végrehajtásával érhetjük el. A motor bekapcsolt állapotban folyamatosan berreg:



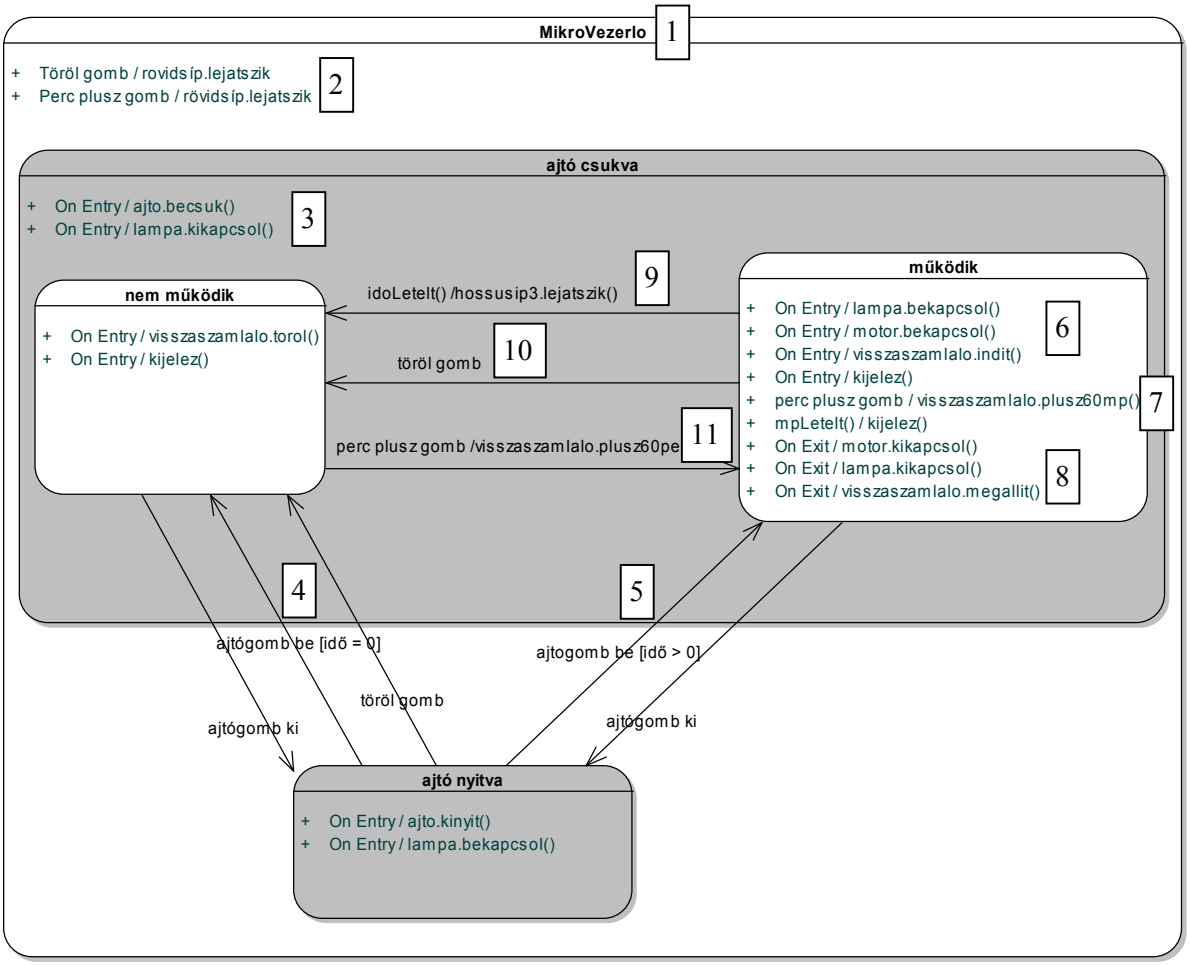
8.5. Egymásba ágyazott állapotok

Az állapotok, illetve az állapotátmeneti diagramok egymásba ágyazhatók. A külső állapot minden tartalmazott állapotra érvényes. Összetett állapotokat az állapotok tartalmazásával tudunk ábrázolni. Az átmenet indulhat akár egy külső, akár egy belső állapotból.

 PÉLDA

A mikrohullámú sütőben két osztály van, melynek érdemes tanulmányozni az állapot-átmeneteit, ezek a Viszaszámláló és a MikroVezérlő osztály. A többi osztály állapotátmenetei triviálisak.

A MikroVezérlő osztály állapotátmeneti diagramját a következő ábra mutatja.



A legfelső szinten maga a MikroVezérlő áll [1], ami azt jelenti, hogy a Mikro mindig ebben az állapotban van. Ha tehát bármikor megnyomjuk a töröl gombot vagy a perc plusz gombot [2], meghívásra kerül a rovidsip.lejatszik().

A mikrónak két állapota lehetséges: *ajtó nyitva* és *ajtó csukva*. Az *ajtó csukva* állapotnak két alállapota van: *motor kikapcsolva* és *motor bekapcsolva*. A tartalmazási sorrendet meg lehetett volna cserélni,

ebben az esetben a motor kikapcsolva állapot tartalmazná az *ajtó nyitva* és *ajtó csukva* állapotot – *motor bekapcsolva* állapotban ugyanis egyértelmű, hogy csukva van az ajtó.

Bárhonnan is érkezünk az *ajtó csukva* állapotba, az ajtót ténylegesen be kell csukni, és a lámpát ki kell kapcsolni [3].

Az *ajtó nyitva* állapotból kétféleképpen juthatunk az *ajtó csukva* állapotba. Mindkét esetben benyomjuk az ajtógombot, de ha idő=0 [4], akkor a *motor kikapcsolva* alállapotba, ha idő>0 [5], akkor pedig a *motor bekapcsolva* alállapotba kerülünk (vagyis ha a visszaszámláló függőben volt, akkor az ajtócsukásra újra elkezd működni a mikró).

Bárhonnan is érkezünk a *motor bekapcsolva* állapotba [6], a lámpát és motort be kell kapcsolni, visszaszámlálót el kell indítani, valamint az időt ki kell jelezni.

Ha a *motor bekapcsolva* állapotban megnyomjuk a perc plusz gombot, a visszaszámláló ideje 60 mp-cel megnövekszik. Ha egy másodperc letelt, akkor az időt ki kell jelezni [7].

Amikor a mikró abbahagyja a működést, akkor a motort és a lámpát ki kell kapcsolni, a visszaszámlálót pedig le kell állítani [8].

Ha az idő letelt, akkor a *motor bekapcsolva* állapotból átkerülünk a *motor kikapcsolva* állapotba, amellett, hogy az ajtó csukva van [9].

Ha *motor bekapcsolva* állapotban lenyomják a töröl gombot [10], akkor *motor kikapcsolva* állapotba kerülünk, a visszaszámláló nullázódik, és az idő kijelzésre kerül.

Ha a *motor kikapcsolva* állapotban lenyomják a perc plusz gombot [11], akkor bejutunk a *motor bekapcsolva* állapotba, miközben a visszaszámláló egy percre áll.

8.6. Kapcsolat a forráskóddal

Az osztály metódusait főként a szekvenciadiagramok alapján határozzuk meg. Ha az állapotváltozások figyelésénél kiderül, hogy hiányzik egy metódus, akkor az osztályt természetesen bővítjük. Az állapotdiagram nagy segítséget jelent az osztály adattagjainak és metódusainak meghatározásában, valamint az egyes metódusok kódolásában.

Állapotátmeneti diagramot főként akkor használunk, ha sok az aszinkron esemény. Az eseménylekezelő metódusok például aszinkron módon kerülnek meghívásra. Ha tehát események létrehozásával főként állapotváltozásokat idézünk elő, akkor az objektum viselkedésének meghatározásához az állapotátmeneti diagram a leghatásosabb módszer.

A kiváltó esemény (trigger) lekezelőjének forráskódjában a következő tevékenységek szerepelnek, ebben a sorrendben:

- az esemény forrásállapotának On Exit tevékenységei;
- a trigger kísérő eseménye;
- a célállapot On Entry tevékenységei;

Ha az adott trigger két helyen is előfordul, akkor a megfelelő tevékenységeket integrálni kell, szükség esetén az állapotok feltételének megfogalmazásával.

 PÉLDA

idő letelt lekezelőjének forráskódja

Az *idő letelt* trigger egyetlen helyen szerepel: a *motor bekapcsolva → motor kikapcsolva* átmenetnél. A *motor bekapcsolva* végén van On Exit:

```
motor.kikapcsol()
lampa.kikapcsol()
visszaszamlalo.megallit()
```

Kisérő esemény:

```
hosszusip3.lejatszik()
```

A *motor kikapcsolva* elején van On Entry:

```
visszaszamlalo.torol()
kijelez()
```

Összegezve:

```
motor.kikapcsol()
lampa.kikapcsol()
visszaszamlalo.megallit()
hosszusip3.lejatszik()
visszaszamlalo.torol()
kijelez()
```

A kódot optimalizásakor a visszaszamlalo.megallit() törölhető.

töröl gomb lekezelőjének forráskódja

A *töröl gomb* trigger három helyen szerepel: legfelső szinten [2], és az *ajtó nyitva → motor kikapcsolva*, valamint a *motor bekapcsolva → motor kikapcsolva* átmeneteknél.

Legfelső szint:

```
rovidsip.lejatszik()
```

A *motor bekapcsolva* esetében van On Exit:

```
if (motor.bekapcsolva) {
  motor.kikapcsol()
  lampa.kikapcsol()
  visszaszamlalo.megallit()
}
```

Kísérő akciók nincsenek.

A *motor kikapcsolva* esetén van On Entry:

```
visszaszamlalo.torol()
kijelez()
```

Összegezve:

```
rovidsip.lejatszik()
if (motor.bekapcsolva) {
  motor.kikapcsol()
  lampa.kikapcsol()
  visszaszamlalo.megallit()
}
visszaszamlalo.torol()
kijelez()
```

ajtógomb lekezelőjének forráskódja

A sütőn egyetlen kétállású ajtó gomb van – ezt le kell kérdeznünk, hogy most milyen állapotba került. Ha a gombbal éppen csukunk, akkor az *ajtó nyitva* állapotból az *ajtó csukva* állapotba; ha éppen nyitunk, akkor az *ajtó csukva* állapotból az *ajtó nyitva* állapotba kerülünk.

Az ajtógomb lekezelésének Java kódja:

```
/* Ajtógombot lenyomták. */
void btAjto_actionPerformed(ActionEvent e) {
  if (btAjto.isSelected()) {

    // ajtócsukás, On Entry:
    ajto.becsuk();
    lampa.kikapcsol();



    if (!visszaszamlalo.nulla()) { // idő > 0
      // motor bekapcsolva On Entry:
      lampa.bekapcsol();
      motor.bekapcsol();
      visszaszamlalo.indit();
      kijelez();
    }
    else {
      // motor kikapcsolva On Entry:
      visszaszamlalo.torol();
      kijelez();
    }
  }
  else {
    // ajtónyitás:
    if (motor.isBekapcsolva()) {
      // motor bekapcsolva On Exit:
      motor.kikapcsol();
      lampa.kikapcsol();
      visszaszamlalo.megallit();
    }
    // ajtó nyitva On Entry:
    ajto.kinyit();
    lampa.bekapcsol();
  }
}
```

9. Aktivitásdiagram

9.1. Az aktivitásdiagram szerepe a modellezésben

Az aktivitásdiagram lényegében egy folyamatábra, segítségével a program dinamikáját ábrázolhatjuk. A diagram megmutatja, hogy az egyes tevékenységek egymáshoz képest mikor, és milyen feltételekkel hajtódnak végre. Az aktivitásdiagram tevékenységei több osztály felelősségi köréhez is tartozhatnak.

Az aktivitásdiagram használható:

- üzleti modellezésre: ekkor a tevékenység lehet az ember vagy a számítógép által elvégzendő tetszőleges tevékenység;
- programspecifikálásra: ekkor a tevékenység általában egy operáció, illetve metódus.

Az aktivitásdiagram az állapotdiagram egy speciális esete, ahol az állapot tevékenység (tevékenység-állapot), a kiváltó esemény (trigger) pedig a tevékenység befejezése.

Az aktivitásdiagram általában a bonyolultabb használati esetek vagy metódusok implementációjához kötődik.

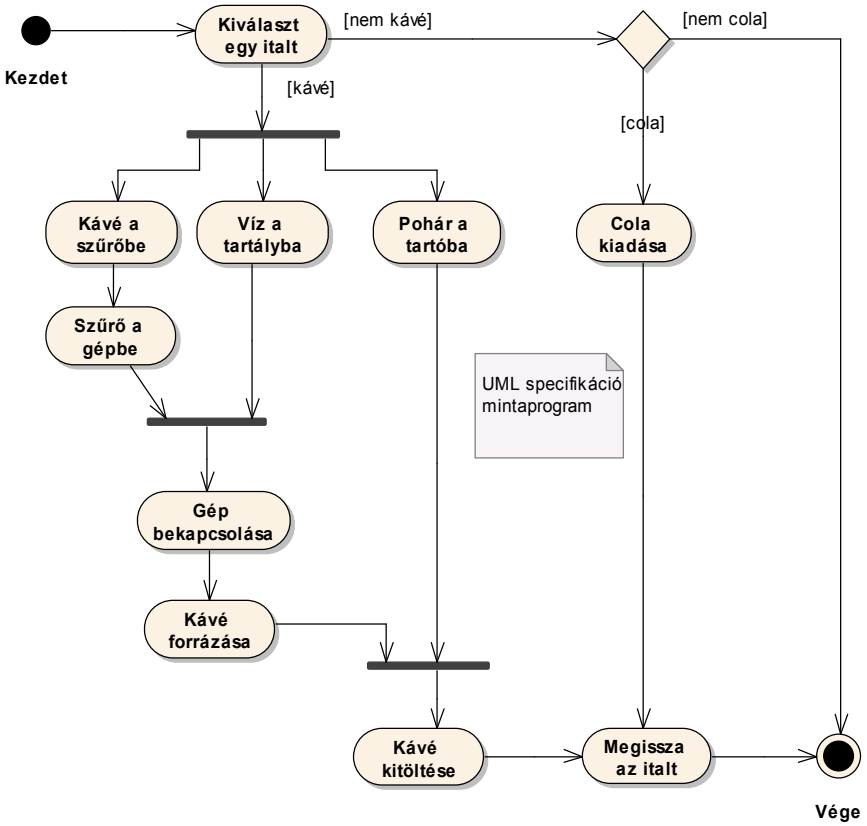
Az aktivitásdiagram a folyamat állapot-gépezete.

9.2. Minta diagramok

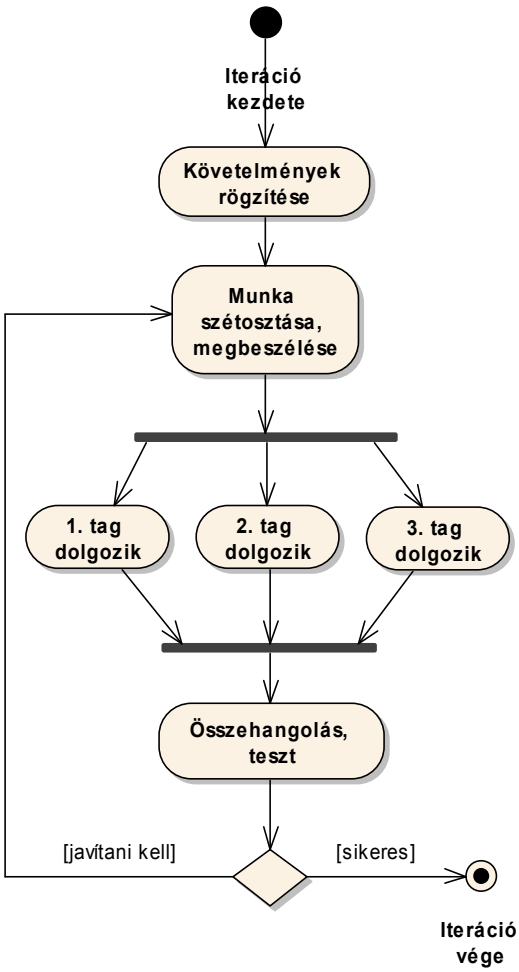
A lekerekített téglalapok tevékenységeket ábrázolnak. A fekete vízszintes vonal az ún. szinkronizációs vonal, amely a hozzá beérkező tevékenységeket „összevárja”. Az élére állított négyzet döntést jelent.

Először tekintsük az UML specifikációban megadott minta aktivitásdiagramot. Az (ingyen működő) italautomatában kétféle ital van: kávé és cola. A használó kiválaszt egy italt. Ha kávét választott, akkor beindul párhuzamosan három tevékenység: 1. kávé kerül a szűrőbe, majd a szűrő bekerül a gépbe; 2. víz kerül a tartályba; 3. egy pohár kerül a tartóba. Ha az első két tevékenységnek vége van, akkor a gép bekapcsolása, majd a kávé forrázása következik. Ha ennek vége van, és a pohár is már benne van a tartóban, akkor kitöltésre kerül a kávé, és meg lehet inni az italt. Ha nem a kávét választották, akkor két eset lehetséges: vagy a colát választották, vagy semmit. Cola esetében a gép kiadja a kért colát, amit máris lehet inni, egyébként nem történik semmi. A folyamatnak vége.

Második példaként nézzük meg a "Szoftverfejlesztés csapatmunkában" egy iterációját! Először rögzítik a követelményeket, majd szétosztják a munkát. Ezután a csapattagok párhuzamosan dolgoznak, összevárják egymást, majd letesztelik a munkákat. Ha még javítani kell, akkor visszamennek a munka szétosztásához. Addig gyúrják a munkát, míg az iteráció sikeresen befejeződik.



Italautomata (UML specifikáció példája)



A "Szoftverfejlesztés csapatmunkában" egy iterációja

9.3. A diagram modellelemei

Kezdet, vég

Egy tevékenységsornak van

- pontosan egy **kezdet**e (belépési pontja); jele egy tömör kör.
- valahány **vég**e (kilépési pontja); jele egy ökörszem.

Tevékenységállapot (action state)

A **tevékenységállapot** (vagy egyszerűen tevékenység) egy olyan állapot, amely maga a tevékenység végrehajtása. A tevékenység befejeződése automatikus állapot-átmenetet vált ki.

A tevékenységállapotnak van egy vagy több bemeneti és egy vagy több kimeneti állapot-átmenete. A bemenetnek van egy belépési eseménye. A kimenet kiváltó eseménye a belépési esemény befejeződése. Több kimenet estén minden átmenetnek további feltétele van (őrfeltétel).

A tevékenységet megadhatjuk természetes nyelven, pszeudokóddal, vagy egy konkrét programnyelvi utasítással.

A tevékenységállapot jele ugyanaz, mint az állapotdiagramon az állapoté; a lekerekített téglalapba itt a tevékenységet írjuk:

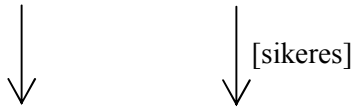
A tevékenységállapot részletezhető, melynek több módja is lehetséges:

- A tevékenységbe alállapotok tehetők;
- A tevékenység modellelem alatt szerepelhet egy újabb diagram, amely az adott összetett tevékenységet részletezi.

Automatikus átmenet (tranzakció)

Aktivitásdiagramnál az egyik tevékenységből a másikba való állapot-átmenet általában automatikus, ekkor a nyílon nincs őrfeltétel. Több átmenet esetén az átmenet természetesen tartalmazhat őrfeltételt (az átmenetek ilyenkor döntéscsomópontból indulnak).

Az átmenet jele:



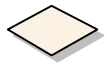
Csomópont (elágazási pont, gyűjtőpont)

Az állapotdiagramon egy állapotból több átmenet is kivezethet; az állapot-átmenetekre kiváltó esemény, őrfeltétel és akció tehető.

A **csomópont** egy alternatív lehetőség az elágazás jelölésére, melyet elsősorban az aktivitásdiagramokon szoktak alkalmazni. A csomópont kétféle lehet:

- **elágazási pont** vagy döntés: egyetlen bemenete, és egy vagy több kimenete van.
- **gyűjtőpont**: egy vagy több kimenete, és egyetlen kimenete van. A gyűjtőpontnak nincs kiváltó eseménye.

A csomópont jele egy élére állított rombusz:



Swimlane (úszósáv)

Az aktivitásdiagram modellelemei felelősségi sávokba rendszerezhetők. Az egyes sávoknak olyan nevet adunk, amely a sávba tartozó tevékenységekre jellemző. Az úszósáv tehát egy adott kontextust jelöl. Ha az egyes sávok objektumokat jelölnek, akkor a sávba tartozó tevékenységeket a megadott objektum végzi. Üzleti folyamat modellezésnél a sáv jelölheti például egy cég szervezeti egységét.

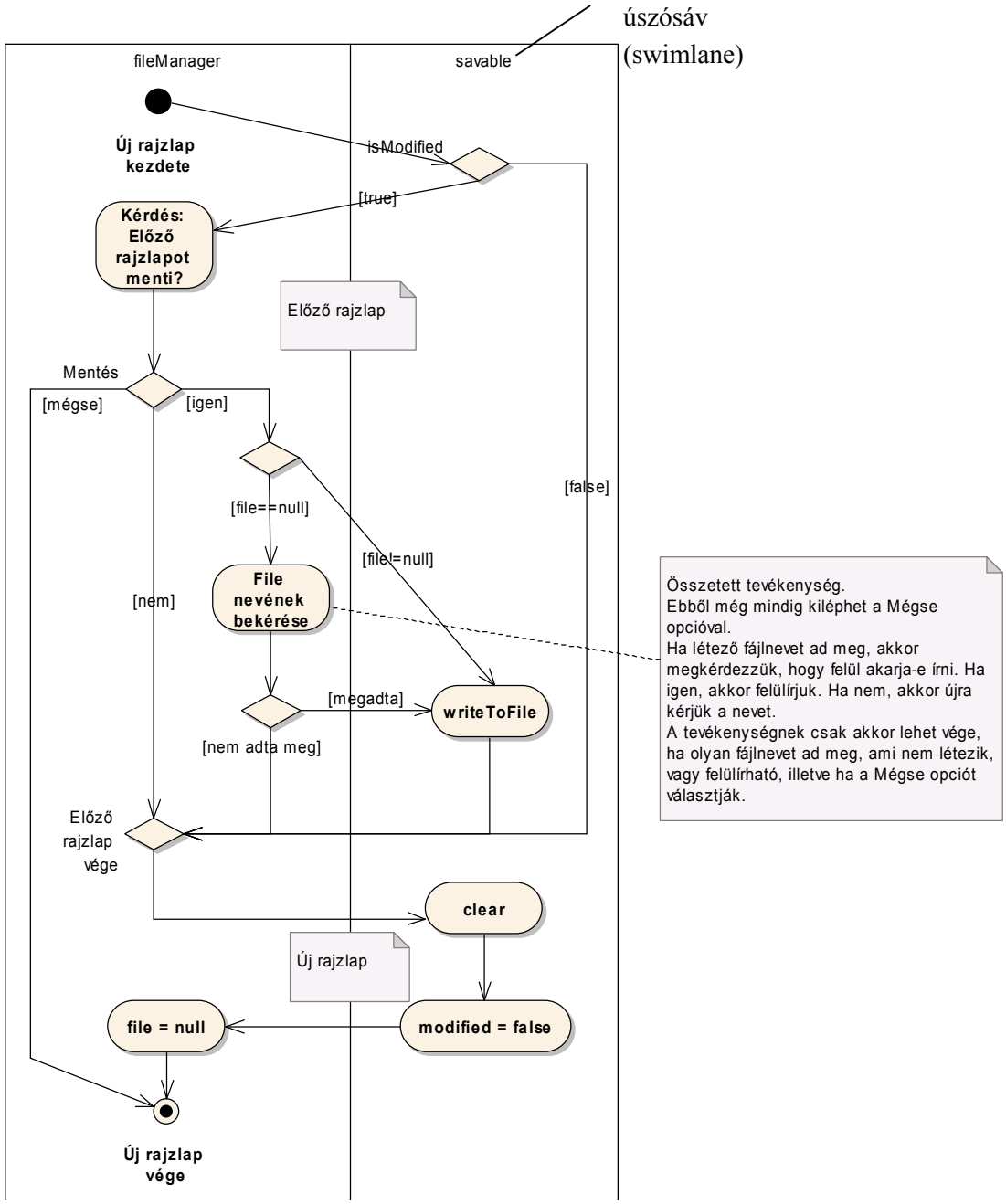
Az úszósávok függőlegesek, azokat vonalak választják el egymástól. A sávok sorrendjének nincs jelentősége.

A KissDraw alkalmazás *Új rajzlap* használati esetének aktivitásdiagramját az ábra mutatja. A tevékenységeket két sávra osztottuk. Az első sáv a fileManager objektum felelősségi köre, míg a második sávba sorolt összes tevékenységet az elmentendő (savable: elmenthető) objektum végzi. A tevékenységsor függőlegesen két részre oszlik: az előző rajzlap mentését az új rajzlap létrehozása követi. A kettőt egy jól látható csomópont (*Előző rajzlap vége*) választja el egymástól – itt futnak össze az *Előző rajzlap* szálai.

Az *Új rajzlap* kiválasztása azzal indul, hogy a fileManager megkérdezi az elmentendő (savable) objektumtól, hogy módosították-e. Ha nem, akkor vége is van az *Előző rajzlap* résznek. Ha igen (isModified==true), akkor megkérdezzük a felhasználót: „Az előző rajzlapot menti?”. Három út lehetséges:

- igen: Ekkor megvizsgáljuk: file==null?, vagyis hogy van-e neve az éppen szerkesztett rajzlapnak (null esetén a file neve Noname). true esetén bekérjük a fájl nevét. Ha megadja, mentünk, ha nem adja, nem mentünk. Ha már volt neve a fájlnak, akkor mindenképpen mentünk (writeToFile).
- nem: Nincs mentés, és ezzel vége is van az előző rajzlap tevékenységsornak.
- mégse: Nem csak az előző rajzlap tevékenységsornak van vége, hanem az *Új rajzlap* tevékenységsornak is. A felhasználó ugyanis meggondolta magát.

Miután vége van az *Előző rajzlap* tevékenységsornak, létrehozzuk az új rajzlapot. Először a savable objektum törli magát, és a modified változóját false-ra állítja. A vezérlés visszakerül a fileManagerhez, aki a file értékét null-ra állítja (a szerkesztendő rajzlap Noname lesz). Ezzel véget ér a használati eset.



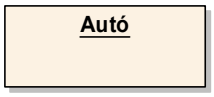
A KissDraw *Új rajzlap* használati esetének aktivitásdiagramja

Megjegyzés:
Egy használati eset viselkedésének modellezéséhez mindig olyan diagramot (szekvencia, együttműködési, aktivitás vagy állapot) használjunk, amely arra a legalkalmasabb!

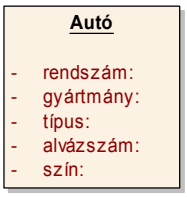
Objektum, objektumfolyam (object flow)

A tevékenységek objetumokban és objektumokon tevékenykednek. Egy tevékenységnek lehetnek objektum bemenetei (input), illetve objektum kimenetei (output). Ezeket az objektumokat az aktivitásdiagramon objektum modellelemekkel ábrázolhatjuk. Az objektum és tevékenység között objektumfolyam kapcsolat van.

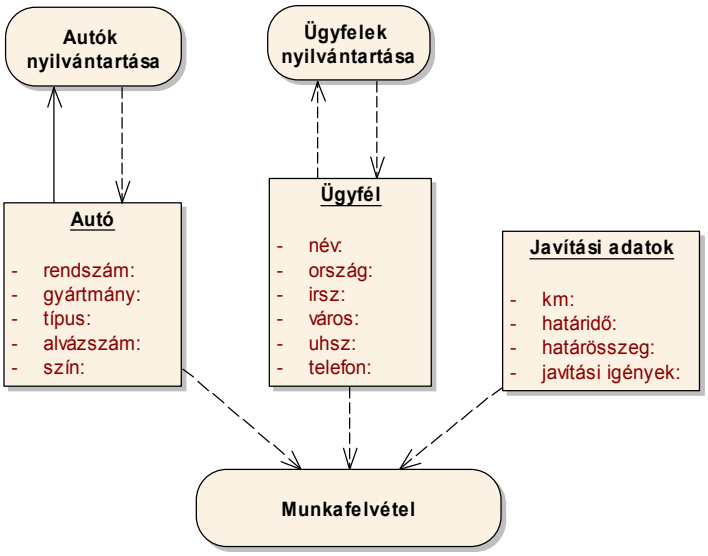
Objektum jele:



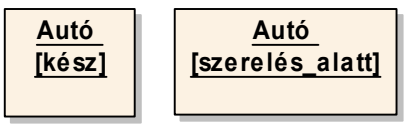
Az objektumba tulajdonságok (és metódusok) tehetők:



Az objektumfolyam jele egy szaggatott nyíl, mely kifejezi az áramlás irányát. Egy tevékenység output objektuma egy vagy több másik tevékenység input objektuma lehet:



Egy objektum többször is rátehető a diagramra. Az objektumok megkülönböztetésére megadható az objektum állapota:

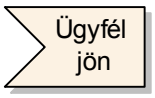


9.4. Események (küldő és fogadott)

Az aktivitásdiagramon lehetőség van események (jelek) elhelyezésére. Az **esemény** egy történés, egy értesítés vagy egy időpont eljövetele. Az esemény elindítja a tevékenységet vagy folyamatot. Az eseményt a tevékenység (folyamat) felhasználhatja, átalakíthatja vagy továbbíthatja. Az esemény lehet küldött esemény (sending) vagy fogadott esemény (receipt).

A küldött esemény jele:

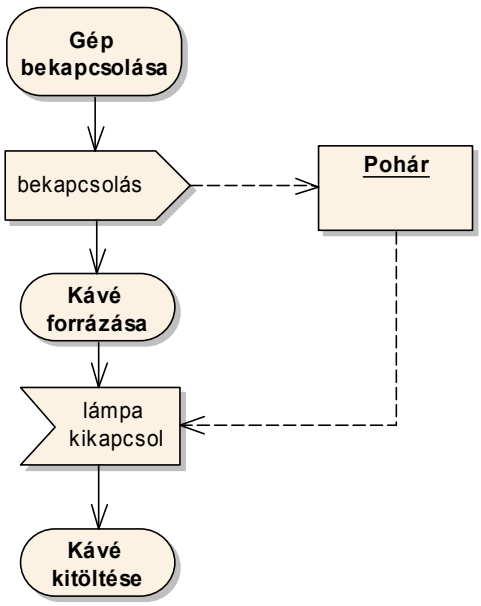
A fogadott esemény jele:



A folyamat által indukált, küldött eseményt egy másik folyamat fogadhatja. Ebben az esetben bármelyik szimbólum használható.

Az esemény jelölésére elvileg nem lenne szükség, mert az átmenet ki tudja fejezni azt. Az esemény jele a fejlesztői közösség nyomására került az UML specifikációba. Ha használjuk, akkor az átmenetnek nincs címkéje, arra nem tehető további esemény.

Egy példa az UML specifikációból, mely az italautomata működésének egy részlete:



A Pohár osztályú objektum a „bekapcsolás” esemény fogadó objektuma, és egyúttal a „lámpa kikapcsol” esemény küldő objektuma is.

9.5. Állapot- vagy aktivitásdiagram?

Az aktivitásdiagram egy osztályszerű modellelemhez kapcsolható, mint használati eset, csomag vagy operáció. A diagramot akkor szokás használni, ha a tevékenységeket belső, szinkronizált események vezérlik, vagyis a programrészlet eljárásorientált. A külső (aszinkron) események által meghatározott programrészletet állapotdiagrammal ábrázoljuk.

Más szóval: az állapotdiagramot eseményvezérelt, az aktivitásdiagramot pedig algoritmusvezérelt programrészletek ábrázolására használjuk.

Az aktivitásdiagram és az állapotdiagram elemei keverhetők. Vegyes elemek esetén a diagram neve (aktivitás- vagy állapotdiagram) attól függ, hogy mely elemek határozzák meg a diagram jellegét.

9.6. Üzleti folyamat modellezés

Az aktivitásdiagram egyik gyakori alkalmazái területe az üzleti folyamat modellezés.

10. Komponens diagram

10.1. A komponensdiagram szerepe a modellezésben

A **komponensdiagram** a rendszert alkotó fizikai komponenseket (szoftverelemeket) és az azok közti kapcsolatokat ábrázola. A komponensdiagramon megadható a logikai nézet osztályainak forráskomponensekhez való hozzárendelése, valamint a forráskódok hozzárendelése futtatható komponensekhez. A komponensdiagram az implementációs nézet jellemző diagramja.

A komponensdiagram a komponensmodell diagramja. A komponensdiagram segítségével rendszerezhetjük, csoportosíthatjuk a rendszer szoftver elemeit, valamint az egyes komponenseket egymáshoz rendelhetjük, egymásba leképezhetjük. Például osztályhoz forráskódot, forráskódhoz futtatható fájlt rendelhetünk.

A komponensdiagramot a következő dolgok modellezésére használják leggyakrabban:

- Forráskódok (java, pas, cpp...)
- Futtatható szoftverelemek (exe, jar...)
- Fizikai adatbázisok (mdb, jds, myd...)

10.2. A diagram elemei

A komponensdiagram modellelemei:

- Komponensek
- Interfészek
- Kapcsolatok: függőség (fordítás, nyomkövetés), öröklés, társítás és megvalósítás.

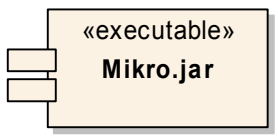
Komponens

A **komponens** (component) egy fizikailag bonthatatlan szoftver egység. A komponens lehet egy állomány, például forráskód, szerkesztendő vagy futtatható szoftver elem: lefordított tárgykód, bájtkód, futtatható program vagy dinamikus könyvtár.

Mint más diagramelemek, a komponensek is csomagokba csoportosíthatók.

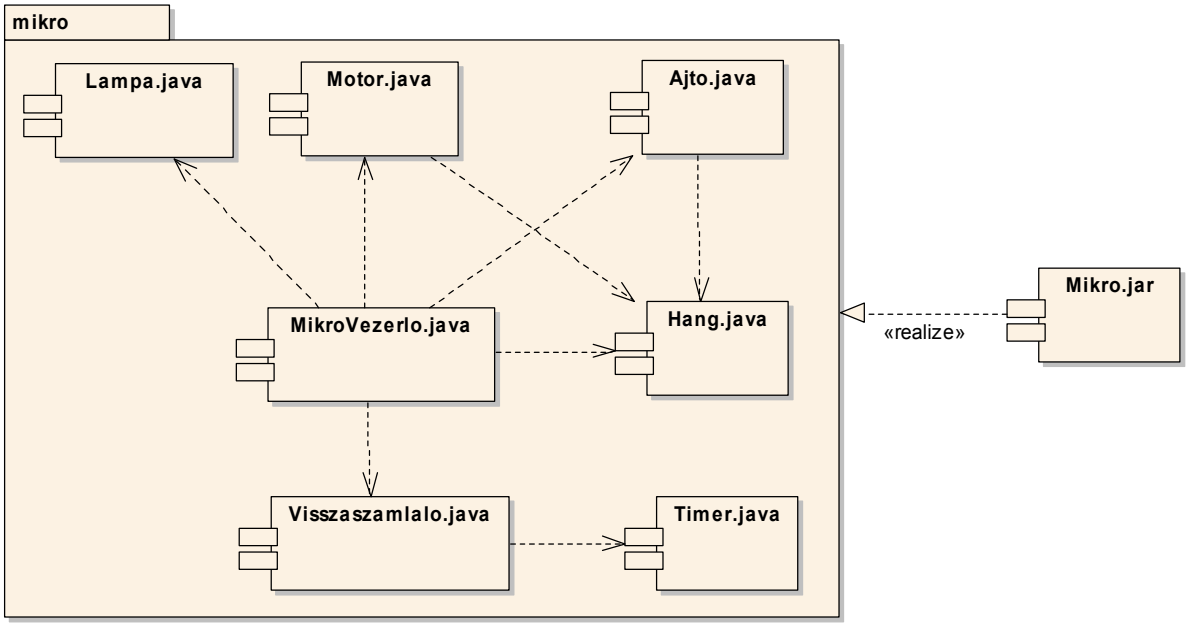
A komponens előredefiniált szereotípusai: executable (futtatható fájl), library (könyvtár), document (dokumentum), table (adattábla), file (fájl), forráskód, kép stb.

A komponens jele:



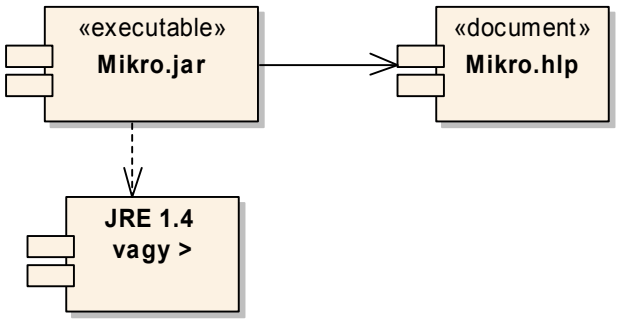
10.3. Minta diagram

A Mikro alkalmazás osztályainak mindegyikéhez egy-egy Java forráskód készül. A következő komponensdiagram a forráskódokat és a közöttük levő függőségi kapcsolatokat ábrázolja. A Java forráskódokat végül is a Mikro.jar realizálja (ez tartalmazza a java fájlokból lefordított class fájlokat).



A Mikro alkalmazás futtatásához a számítógépen három dolognak kell jelen lennie:

- Mikro.jar: Futtatható class fájlok és tartozékok együttese.
- Mikro.hlp: Help dokumentum. A Mikro.jar használja.
- JRE 1.4 futtató környezet és könyvtár.



11. Telepítési diagram

11.1. A telepítési diagram szerepe a modellezésben

A **telepítési diagram** a telepítési modell diagramja. Itt adjuk meg a rendszer topológiáját. A telepítési diagram segítségével a rendszer szoftver elemeit a hardver elemekhez rendeljük. Megadjuk, hogy mely komponensek (szoftver elemek) milyen számítógépen futnak. A telepítési diagram az implementációs nézet jellemző diagramja.

11.2. A diagram elemei

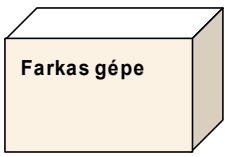
A telepítési diagram modellelemei:

- **Csomópont:** egy hardver elem, processzor, számítógép vagy egyéb eszköz. A csomópont szerotípusai például: pc, pc-kliens, pc-szerver, printer, cd-rom, storage (tároló) stb.
- **Kapcsolat:** A csomópontok közötti kapcsolat lehet társítási, öröklési, tartalmazási stb. A kapcsolat sztereotípusa lehet például: lokális háló, TCP/IP.
- **Komponens:** A csomópontokra komponensek tehetők. Ez jelzi, hogy a komponens a hardver elemen van, illetve fut.

Csomópont (node)

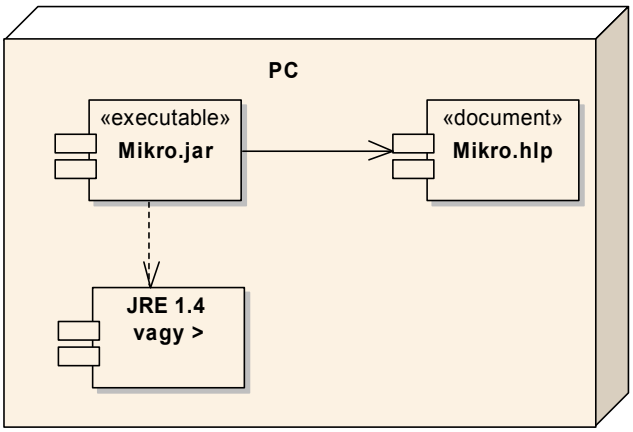
A **csomópont** egy számítógép vagy hardver eszköz, mely csak a rendszer futásakor létezik, és általában van memóriája vagy adatfeldolgozó képessége. Csomópont például egy konkrét számítógép (szerver vagy kliens), egy nyomtató vagy egy levilágító.

A csomópont jele:



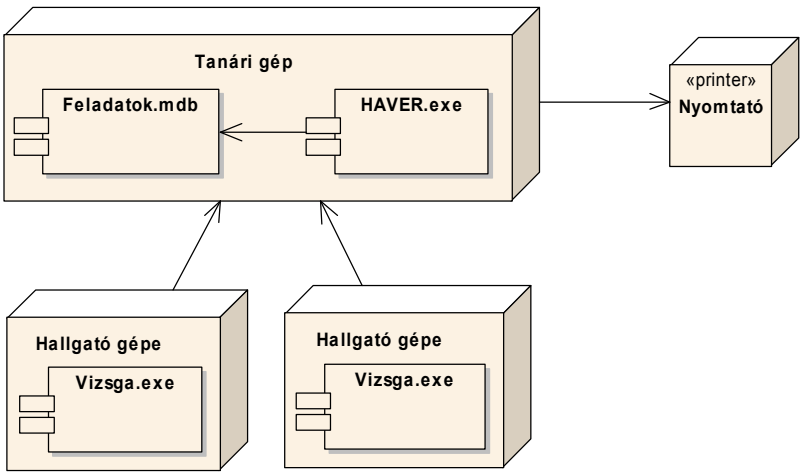
11.3. Minta diagramok

A Mikrohullámú sütő telepítési diagramja triviális. Bármely PC-n használható, amelyre telepítik. A Mikro.jar programot kell a gépre másolni. A futás feltétele, hogy jelen legyen a JRE és a help állomány.



A mikrohullámú sütő telepítési diagramja

A HAVER (Hallgatói Vizsgáztató és Ellenőrző Rendszer) telepítési diagramja már egy fokkal összetettebb. Itt más dolgok vannak a tanári gépen és a hallgatók gépén. A tanári gépen jelen van a teljes HAVER szoftver, mely segítségével feladatok is készíthetők. A hallgatói gépen csak a Vizsga.exe fut, amely azonban eléri a tanár gépén levő feladatokat. A tanári géphez egy nyomtatóva kötve.



A HAVER telepítési diagramja

Irodalomjegyzék

[1] OMG Unified Modeling Language Specification. Version 1.4, 2001

[2]