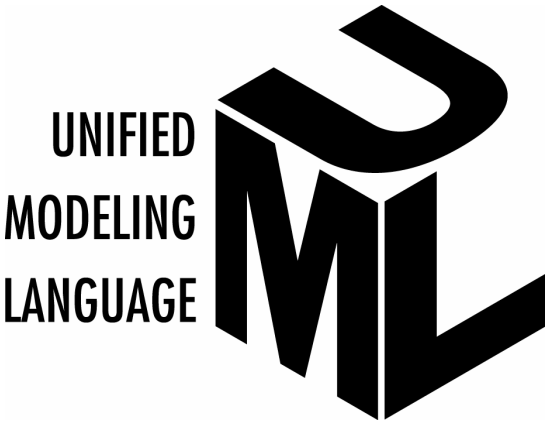
DIPLOMAMUNKA

BENYÁCS TAMÁS

BUDAPEST

1998

GÁBOR DÉNES
FŐISKOLA



az uml objektum-orientált modellező nyelv szemantikája

539/1998

BUDAPEST

1998

Tartalomjegyzék

ábrajegyzék

irodalomjegyzék

[1] Booch, Grady - Jacobson, Ivar - Rumbaugh, James: Unified Modeling Language Semantics; version 1.1 (www.rational.com, 1 September 1997)

[2] Angster Erzsébet: Az Objektum Orientált tervezés és programozás alapjai (Angster Erzsébet, 1998)

1. BEVEZETÉS

1 Előszó

Az UML (Unified Modeling Language=Egységes Modellező Nyelv) a szoftverrendszerek résztermékeinek meghatározására, szemléltetésére, összeállítására és dokumentálására kifejlesztt nyelv, mely a szoftver iparágon kívül egyéb területeken, pl. az üzleti modellezésben is kiválóan alkalmazható. Az UML egyszerű és hatékony. A nyelv az alapfogalmak egy igen szűk halmazán alapszik, melyeket a legtöbb objektum-orientált fejlesztő könnyen elsajátíthat és alkalmazhat. Az alapfogalmak kombinálhatók és kiterjeszthetők, így az objektum modellező szakemberek igen terjedelmes és összetett rendszereket képesek definiálni a legkülönfélébb alkalmazási területeken. A szakdolgozat célja e módszertan szemantikájának bemutatása a magyar szoftvertervező szakemberek számára.[1]

Legelőször is az UML a Booch, az OMT (Object Modeling Technique=Objektum Modellezési Módszer) és OOSE (Object Oriented System Engineering=Objektutm Orientált Rendszertervezés) koncepcióit olvasztja egybe. Az eredmény egy egyedülálló, közös és széles körben használható modellezési nyelv ez előbbi, illetve más módszerek használói számára[2].

Másodszor, az UML átöleli mindazt, ami elvégezhető a korábbi módszerekkel. Példaként az UML szerzők célbavették a konkurens, elosztott rendszerek modellezését, hogy meggyőződjenek arról, hogy az UML megfelelően alkalmazható e területen is.

Harmadszor, az UML egy szabványos modellezési nyelvre irányítja a figyelmet, nem egy szabványos folyamatra. A különböző szervezetek és problématerületek különböző folyamatokat igényelnek, ezért az UML kidolgozói erőfeszítéseiket egy közös metamodell (mely egységesíti a szemantikát) megalkotására és egy közös jelölésrendszerre összpontosították (mely e szemantikának egy jól értelmezhető formát kölcsönöz).

Az UML olyan modellezési nyelvet specifikál, mely az OO közösség által elfogadott modellezési alapfogalmakat foglalja magába. A kiterjesztési mechanizmus által természetesen

[1] A dolgozat alapját képező angol nyelvű "UML Semantics v 1.1" c. dokumentum az Interneten a www.rational.com WEB cím alatt megtalálható.

[2] Az UML - bizonyos absztrakciós szinten - jó eszköz lehet más - nem műszaki - tudományágak képviselői számára is: pl a társadalmi forlyamatok, mint amilyen az egyes politikai pártok hatalmi viszonyainak, népszerűségének alakulása az esetleges gazdasági folyamatok függvényében; vagy az imént említett gazdasági folyamatok alakulása egy adott EU országban az EU integrációs folyamat függvényében; vagy pl. jó eszköz lehet az anyagszerkezettani modellezésekhez, biológiai folyamatok (pl. populáció, immunrendszer, genetika) modellezéséhez. Műszaki tudományágon belül megemlíthető egy olyan komplett zenei rendszer tervezése, modellezése, melyben az elektronikus zenei egységek - mint objektumok - egy meghatározott szabványos kommunikációs csatornán (MIDI) keresztül kommunikálnak egymással, illetve egy számítógéppel, mely ezek koordinációját, felügyeletét látja el, s mely esetleg on line WEB szolgáltató is lehet egyben: bemenetként egy kívánt zenemű azonosítóját kéri be, mely lehet akár egy - a zeneműben előforduló - zenei motívum bejátszása, a kimenete pedig a zeneművet reprezentáló MIDI fájl, melyből a kliens oldalon egy kapcsolódó célprogram által azonnal generálható a kotta is. E zenei rendszer színpadi alkalmazásban alkalmas lehet pl. egy kórus hangszeres kíséretére; ez esetben a kórussal való szinkronról a karmesteri pálca végén elhelyezett infra led által kibocsájtott, és a rendszer infra érzékelője által MIDI - órajelként feldolgozott jelek gondoskodnának. Ez utóbbi terv megvalósítása a dolgozat írójának (közel)jövőbeli célja. (E lábjegyzetben felsoroltak a szakdolgozat írójának magánvéleményét képezik, szakmai vita alapja lehet.)

lehetőség nyílik e fogalmaktól való eltérésre is. Az UML fejlesztői a következő szempontokat tartották szem előtt munkájuk során:

- Elegendő szemantika és jelölés biztosítása a mai igen összetett modellezési problémákhoz gazdaságos és közvetlen formában;
- Elegendő szemantika és jelölés biztosítása bizonyos várható modellezési problémához, különösen a komponens technológiára, elosztott számításra, keretrendszerre, ill. a végrahajthatóságra vonatkozólag.
- Kiterjesztési mechanizmus biztosítása, melynek segítségével az egyes projektek alacsony költségek mellett terjeszthetik ki a metamodellt saját alkalmazásaikhoz.
- Kiterjesztési mechanizmus biztosítása, mely által a jövőbeli modellezések az UML tetőpontjáig teljesedhetnek ki.
- Elegendő szemantika biztosítása az egyes modelleknek a különböző eszközök közötti cserélhetőségéhez.
- Elegendő szemantika biztosítása olyan interfész specifikációjához, mely a modellezési termékek megosztott tárolását teszi lehetővé.

Az UML specifikáció két egymással szoros összefüggésben lévő részből áll:

- *UML Szemantika.* Egy metamodell, mely az UML objektum modellezési fogalmak absztrakt szintaxisát és szemantikáját írja le.
- *UML Jelölésrendszer.* Grafikus jelölés az UML szemantikájának vizuális megjelenítéséhez.

Az UML a legkifinomultabb mérnöki praktikák gyűjteményét reprezentálja, melyek igen nagy és összetett rendszerek modellezése során is sikeresnek bizonyultak.

2 Az UML története

Az UML -t a Rational Software és partnerei fejlesztették ki. E nyelv a Booch. OOSE/Jacobson, OMT és más módszerek modellező nyelveinek utóda. Számos cég vezeti be az UML -t mint szabványt a fejlesztéseihez. Az UML olyan társtudományágakat ölel fel, mint az *üzleti modellezés, követelmény menedzsment, elemzés és tervezés, programozás,* valamint *a tesztelés*.

.1 UML 0.8 - 1.1

Az UML előfutárai

Az első azonosítható objektum-orientált modellező nyelvek a ‘70 -es évek közepe és a ‘80 -as évek vége között kezdtek megjelenni, ahogy a különböző módszertannal foglalkozó szakemberek az objektum-orientált elemzéssel- és tervezéssel kísérleteztek különböző megközelítéssel. Számos más technika is befolyásolta e nyelveket, mint pl. az *Egyed-Kapcsolat modellezés, a Specifikációs és Leíró Nyelv* (SDL, sicra 1976, CCITT), és más technikák. 1989 és 1994 között az azonosított modellezési nyelvek száma kb. 10 -ről több, mint 50 -re emelkedett. Az objektum-orientált módszertanok felhasználóinak igen népes

tábora küszködött a kielégítő modellezési nyelv hiányával, "módszertani háború" alakult ki. A '90 -es évek közepétől e módszertanok ismét kezdtek megjelenni. A legfigyelemereméltóbb ezek közül a Booch '93, mely az OMT továbbfejlesztett és fuzionált változata. E módszertanok elkezdték felvenni egymás technikáit, és hamarosan feltűnt néhány kiemelkedően jó módszertan, mint pl. az OOSE, OMT-2 és a Booch '93. Ezek mindegyike egy-egy komplett módszertan volt és mindegyiknek megvolt a maga erőssége. Az OOSE használati-eset orientált megközelítést alkalmazott, mely kiválóan támogatta az üzleti modellezést és a követelmény elemzést. Az OMT-2 különösen kifejező volt az elemzéshez és az intenzív adatkezelő információrendszerek kifejlesztéséhez. A Booch '93 igen alkalmasnak bizonyult a projektek tervezési- és konstrukciós fázisaiban, valamint népszerű volt a mérnöki alkalmazásoknál is.

Booch - Rumbaugh - Jacobson társulás

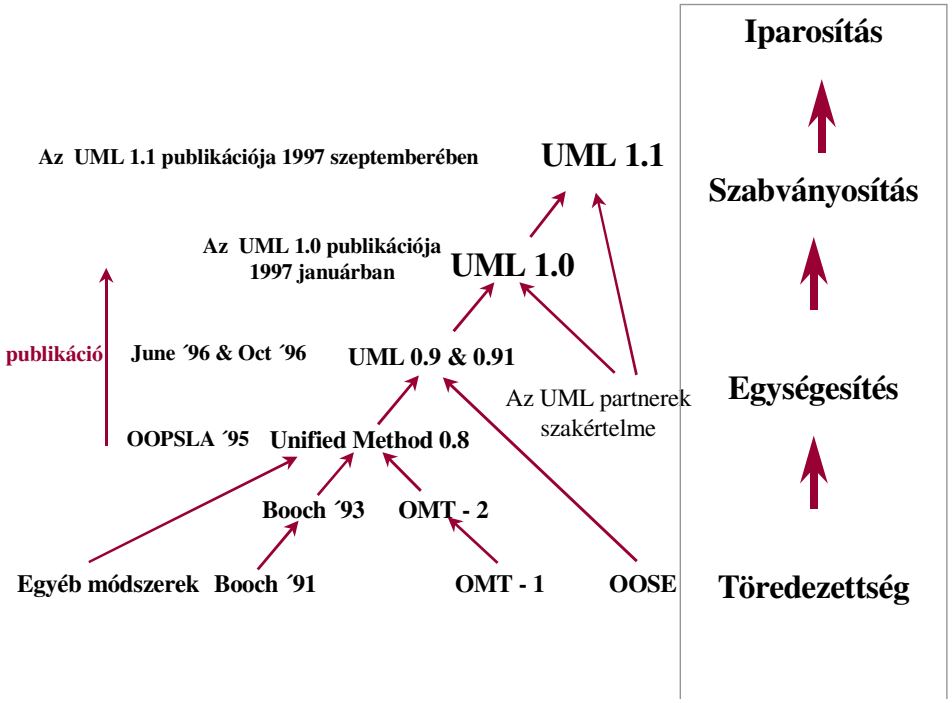
Az UML fejlesztése 1994 októberében kezdődött, amikor Grady Booch és Jim Rumbaugh, a Rational Software Corporation részéről egységesíteni kezdték munkájukat, a Booch és az OMT módszertanokat. Mivel a Booch és az OMT módszertanok függetlenül fejlődtek együtt és az egész világon vezető objektum-orientált módszertanokként ismerték őket, Booch és Rumbaugh egyesítették erőiket, hogy egységesítsék munkájukat. Az "Egységes Módszertan (Unified Method)" - ahogyan akkor hívták - 0.8 verziója 1995 októberében látott napvilágot. 1995 végén Ivar Jacobson és Objectory nevű cége csatlakozott a Rational nevű céghez és egyben az egységesítési törekvéshez. Jacobson az OOSE (Object-Oriented Software Engineering) módszertan fuzionálásával járult hozzá az új egységes nyelv megalkotásához.

Grady Booch, Jim Rumbaugh és Ivar Jacobson, mint a Booch, OMT és az OOSE módszertanok szerzőit három tényező is ösztönözte az egységes modellezési nyelv megalkotásában. Először is, e módszertanok már egyre inkább egymás irányába fejlődtek. Az együttes fejlődéssel elkerülhetők az olyan felesleges eltérések, melyek a felhasználókat megzavarhatják. Másodszor a szemantika és a jelölésmód egységesítése az objektum-orientált piacon való stabilitást is fokozza. Harmadszor a szerzők együttműködésüktől a három korábbi módszertan fejlődését várták. Egymás módszereit megtanulva sokkal hatékonyabban képesek a felmerülő problémákat megoldani, mint előzőleg.

Az egységesítés kezdetén a szerzők az alábbi alapvető célokat fogalmazták meg erőfeszítéseik összpontosítására:

- Rendszerek (nem csak szoftverrendszerek) modellezésének lehetővé tétele az objektum-orientált fogalmak felhasználásával;
- Egyértelmű kapcsolat létesítése a fogalmi és a végrehajtható termékek között;
- Ember és gép által egyaránt jól értelmezhető modellezési nyelv megalkotása.

Booch, Rumbaugh és Jacobson erőfeszítései az UML 0.9 verziójának megjelenését eredményezték 1996 őszén. Ezt követték 1997 -ben az 1.0 és 1.1 verziók.



3 Mi ösztönözte az UML definiálását?

Ez a rész számos tényezőről szól, melyek az UML megalkotását motiválták. Megvitatjuk, miért is modellezünk lényegében, kiemelve néhány kulcsfontosságú kifejezést a szoftveriparon belül, valamint rávilágítunk azokra a problémákra, melyeket a modellezés megközelítéseinek eltérése okoz.

.1 Miért modellezünk?

Egy modell megalkotása ugyanúgy előfeltétel egy ipari szoftverrendszer kifejlesztéséhez, mint egy tervrajz elkészítése egy nagy építkezéshez. A jó modellek nélkülözhetetlenek a projekt csoportok közötti kommunikációhoz és biztosítják az egységes gondolkodásmódot. Azért építünk modelleket összetett rendszerekhez, mert nem vagyunk képesek teljességében átlátni bármilyen bonyolultságú rendszert. Ahogyan növekszik a rendszerek összetettsége, úgy válnak egyre fontosabbá a jó modellezési technikák. Számos további tényező szükséges egy projekt sikeréhez, de egy szigorú *modellezési nyelv* szabvány a leglényegesebb tényező. A modellező nyelvnek a következőket kell tartalmaznia:

- Modell elemek - alapvető modellezési fogalmak és szemantika;
- Jelölésmód - a modell elemek vizuális megjelenítése;
- Irányelvek - a szakmabeli felhasználás sajátságai.

Az egyre összetettebb rendszerek megjelenésével mind lényegesebbé válik a képi szemléltetés és a modellezés. Az UML egy jól definiált és széles körben elfogadott megoldást nyújt e kihívásra.

.2 Ipari tendenciák a szoftvergyártásban

Mivel a szoftver stratégiai jelentősége egyre fokozottabb a vállalkozások számára, az ipari termelésben alkalmazott módszerek látszanak a legalkalmasabbaknak a szoftverek előállításának automatizálására. Arra keresünk technikákat, hogy javítsuk a minőséget és csökkentsük a költségeket, valamint az elkészült terméknek a kereskedelembeli átfutási idejét. E technikák tartalmazzák a komponens technológiát, a vizuális programozást, mintasablonokat és keretrendszereket. Technikákat keresünk továbbá a rendszerek összetettségének menedzselésre, amint azok működési köre és mérete egyre növekszik. Különösen felismerhetjük a szükségét olyan architekturális problémák megoldásának, mint pl. a fizikai disztribúció, konkurencia, replikáció, biztonság, betöltési kiegyenlítettség, és hibatűrés. (Itt említhetjük példaként, hogy a világháló (World Wide Web) fejlesztése számos dolgot egyszerűbbé tesz, de egyben súlyosbítja is az architekturális problémákat.)
A rendszerek összetettsége az alkalmazási területek és a feldolgozások függvényében változik. Az UML fejlesztőinek egyik kulcsindítéka az volt, hogy létrehozzanak egy olyan szemantikai- és jelölési rendszert, mely megfelelően viselkedik az architekturális összetettség minden szintjén és minden felhasználási területen.

.3 Az ipari irányultság előtt...

Az UML előtt nem létezett egyértelműen elfogadott vezető modellezési nyelv. A felhasználók kénytelenek voltak választani a sok egyszerű modellező nyelv közül. E nyelvek legfőképp kifejezési képességeikben tértek el egymástól. A legtöbb modellező nyelv tartalmazott egy olyan fogalomkört, mely a szakmában általánosan elfogadott, azonban e fogalmakat kissé különböző módon fejezték ki. Az egységes megállapodásnak e hiánya az új felhasználókat elriasztotta az objektum orientált piacra való belépéstől, valamint az objektum orientált modellezéstől. Nagymértékben ki kell tehát terjeszteni a modellezés kifejező erejét, hatékonyságát. A felhasználók régi vágya volt, hogy az ipar alkalmazzon egy olyan, széles körben támogatott modellező nyelvet, mely általános célokra alkalmazható. Egy modellező *"lingua franca"* -ra (egységes, közös nyelvre) vártak.

Néhány kereskedő visszariadt az OO modellezési területről, mert hasonló, de egymástól kissé eltérő modellezési nyelveket kellett támogatni. A sok modellező nyelv támogatása és használata miatt felmerült költségek a vállalatokat az OO technológia alkalmazására és az UML fejlesztésének támogatására sarkallták.

Igaz, hogy az UML önmgában még nem garantálja a projekt sikerét, de alkalmazása számos problémát old meg. Pl. jelentősen alacsonyabbak az oktatási és átállási költségek, amikor váltani kell két projekt vagy szervezet között. Alkalmat nyújt az eszközök, eljárások és területek közötti új integrációhoz. De a legfontosabb, hogy lehetővé teszi a fejlesztők számára, hogy figyelmüket az üzleti érdekekre összpontosíthassák, és paradigmát biztosít számukra a megvalósításhoz.

4 Az UML céljai

Az UML tervezésekor az elsődleges célok a következők voltak:

- Használatra kész, kifejező vizuális modellező nyelv biztosítása a felhasználók számára, hogy jól értelmezhető modelleket készíthessenek, valamint cserélhessenek egymás között.
- Az alapfogalmak kiterjesztése érdekében kiterjesztési és specializációs mechanizmus biztosítása;
- A konkrét programozási nyelvektől és a fejlesztési eljárásoktól való függetlenség;
- Formai alap biztosítása a modellezési nyelv értelmezéséhez;
- Az OO eszközök piacának serkentése;
- Magasabbszintű fejlesztési koncepciók támogatása, mint pl. együttműködés, keretrendszerek, mintasablonok és komponensek;
- A legjobb praktikák integrációja.

5 Az UML rendeltetése

A dolgozat egyik elsődleges célkitűzése, hogy megadja az UML szemantikus felépítésének átfogó, pontos specifikációját. Az UML felhasználói köre kiterjed a szabványügyi szervezetekre, szoftverfejlesztőkre, metamodell készítő, módszertani- és egyéb szakemberekre.

6 Kiterjedés

A dolgozatban sor kerül a struktúrális és viselkedési objektum modellek szemantikájának specifikálására. A *strukturális* modellek (*statikus* modellek) a rendszerbeli objektumok felépítését hangsúlyozzák ki, szemléltetve osztályaikat, interfészeiket, attributumaikat és kapcsolataikat. A *viselkedési* modellek (*dinamikus* modellek) a rendszerbeli objektumok viselkedéseit emelik ki, ábrázolják e modellek metódusait, kölcsönhatásait, együttműködéseit és állapot történeteit.

A specifikáció teljes szemantikát biztosít minden modellezési jelöléshez, mely az *UML jelölésrendszer* dokumentumban megtalálható. Ez a diagram technikák széles köréhez nyújt támogatást: *osztály diagram, objektum diagram, szekvencia diagram, együttműködési diagram, állapot diagram, aktivitás diagram és telepítési diagram*. Az *UML jelölésrendszer* dokumentum[3] azon szemantika-szekciók összefoglalását tartalmazza, melyek a diagram technikákhoz kötődnek.

[3] Külön dokumentum, nem képezi tárgyát e szakdolgozatnak.

7 Megközelítés

A specifikáció kihangsúlyozza a nyelvi felépítést és szigort. Az UML felépítése egy négyszintű metamodell struktúrán alapszik, mely a következő szinteket (rétegeket) tartalmazza: **Felhasználói objektumok, Modell, Metamodell és Meta-metamodell**. A dolgozat elsősorban a metamodell réteggel foglalkozik, mely a meta-metamodell réteg egy példánya. Pl. az *Osztály* a metamodellben a *MetaOsztály* egy példánya a meta-metamodellen belül. Az UML metamodell felépítését a *Nyelvi felépítés* rész tárgyalja részletesebben.

Fontos kihangsúlyozni, hogy az UML metamodell egy logikai, nem pedig fizikai (vagy implementációs) modell. A logikai metamodell előnye, hogy kihangsúlyozza a deklaratív szemantikát, és háttérbe szorítja az implementáció részleteit. A logikai metamodellhez használt implementációknak illeszkedniük kell a szemantikához, és importálhatóknak ill. exportálhatóknak kell lenniük, csakúgy mint a részmodelleknek. Az eszközforgalmazók különböző módon is összeállíthatják a logikai metamodellt, ezért implementációikat a valósághoz hangolhatják. A logikai modell hátránya az, hogy hiányzik belőle a pontos és hatékony implementációhoz szükséges imperatív szemantika. Ennek következtében a metamodellt az eszközkészítők számára készített implementációs jegyzetek kísérik.

Az UML a metamodell szinten belül is struktúrált. A nyelv több logikai csomagra oszlik: **Alap, Viselkedési elemek, Általános mechanizmusok**. E csomagok alcsomagokra bomlanak. Pl. az Alap csomag tartalmazza a Törzs, a Kiegészítő elemek, a Kiegészítő mechanizmusok és az Adattípusok alcsomagjait. A nyelv struktúráját teljes részletességgel tárgyalja a *Nyelvi felépítés* szakasz.

A metamodell leírása félig formális módon, három nézetből történt:

- Absztrakt *szintaxis*;
- Jól képzett *szabályrendszer*;
- *Szemantika*.

Az **Absztrakt Szintaxis** egy modell, mely az UML egy részhalmazában van leírva, egy UML osztály diagramból és egy kiegészítő természetes nyelvi leírásból áll. (Ilymódon az UML önmagát hozza létre, hasonlóan, mint ahogyan egy fordítóprogramot saját maga lefordítására használunk). A **Jól Képzett Szabályrendszer** egy formális nyelv (Object Constraint Language = objektum megszorítások nyelve) és természetes nyelv (angol[4]) által biztosított. Végül a szemantika elsősorban természetes nyelven íródott, de tartalmazhat néhány további jegyzetet a leírt modelltől függően. A nyelvet specifikáló formális technikák adaptációja teljes részletességgel a *Nyelvi formalizmusok* szakaszban olvasható.

Összefoglalva tehát, az UML metamodell leírása a ***grafikus jelölésmód, a természetes nyelv*** és ***a formális nyelv*** kombinációja.

8 A dolgozat felépítése

A dolgozat az alábbi fő részeket tartalmazza:

[4] Jelen esetben magyar

- Első rész: Háttér.
- Második rész: Alap.
- Harmadik rész: Viselkedési elemek.
- Negyedik rész: Általános mechanizmusok.

Az első rész az UML *struktúráját és specifikációját* fejti ki. A nyelvi felépítés szakasz leírja a nyelv struktúráját és részletezi a négyszintű metamodell felépítést. A nyelv specifikációját tartalmazó szakasz több nézetből is leírja a nyelv szigorú definícióját.

A második rész definiálja az *infrastruktúrát* az UML számára, ez az alapcsomag. Az alapcsomag több részcsomagra bomlik: Törzs, Kiegészítő elemek, Kiterjesztési mechanizmusok és Adattípusok. A Törzs, vagyis a központi csomag specifikálja az alapfogalmakat, melyek egy elemi metamodellhez szükségesek és definiál egy architekturális gerincet a további nyelvi konstrukciókhoz, amilyenek pl. a metaosztályok, metaasszociációk, és metaattributumok. A kiegészítő elemek csomagja további konstrukciókat definiál, melyek kiterjesztik a központi csomagot olyan további fogalmak támogatására, mint pl. a függőségek, sablonok, fizikai struktúrák és nézet elemek. A kiterjesztési mechanizmusok csomag specifikálja a modell elemek alkalmazásának és új szemantikával való kiterjesztésének módját. Az adattípusok csomag alap adatstruktúrákat definiál a nyelvhez.

A harmadik rész a *szuperstruktúrákat* definiálja az UML -beli *viselkedés* modellezéshez. A viselkedési elemek csomag négy szintű alcsomagból áll: Általános viselkedés, Együttműködés, Használati esetek, és Állapot gépek. Az Általános viselkedés specifikálja azokat az alapfogalmakat, melyek a viselkedési elemekhez szükségesek. Az Együttműködési csomag specifikál egy viselkedési kontextust a modell elemek használatához, mely egy adott feladat végrehajtásához szükséges. A Használati eset csomag az szereplők (aktorok[5]) és használati esetek felhasználásával viselkedést specifikál. Az Állapot gépek csomag *végállapot átmenet rendszerek* felhasználásával definiál viselkedést.

A negyedik rész a modellek *általános alkalmazhatóságának* mechanizmusait definiálja. Az UML e verziója egy általános mechanizmus csomagot tartalmaz, a *Modell Menedzsmentet*. A Modell Menedzsment csomag azt határozza meg, hogy a modell elemek hogyan szerveződnek modellekbe, csomagokba és rendszerekbe.

[5] A következőkben az érthetőség érdekében szereplőkként hivatkozunk az aktorokra.

2. I. RÉSZ: HÁTTÉR

Az első rész kifejti az UML struktúráját és specifikációját. A nyelvi felépítés szakasz leírja a nyelv struktúráját és elmagyarázza a négyszintű metamodell felépítést. A nyelvi formalizmus szakasz leírja, hogyan definiálták a nyelvet a három kiegészítő nézet felhasználásával.

1 A nyelv felépítése

.1 Áttekintés

Az UML metamodell - az UML jelölésrendszerének és szemantikájának felhasználásával - teljes szemantikát definiál az objektum modellek reprezentációjához. E módon az UML saját magát hozza létre, hasonlóan, mint ahogyan egy fordítóprogramot saját maga lefordítására használunk.

Az UML metamodell a négyszintű metamodell architektúra egyike. Mivel e metamodell szint igen összetett, logikai csomagokra bontjuk. Az UML csomagok növelik a nyelv modularitását és támogatják a többszörös engedélyezési pontokat. A következő szakasz áttekintést nyújt az UML négyszintű metamodell architektúrájáról és ismerteti annak csomag struktúráját.

.2 Négy szintű metamodell felépítés

Amint azt az előző szakaszban is olvashattuk, az UML metamodellt a négyszintű metamodell architektúra egyikeként definiálták. Ez a felépítés egy megfelelő infrastruktúra annak a pontos szemantikának a definiálásához, mely az összetett modellekhez szükséges. Számos más előnnyel is jár e megközelítés:

- Érvényesíti az alapfelépítményeket rekurzív módon alkalmazva azt az egymásra épülő metarétegekhez.
- Architekturális alapot biztosít a jövőbeli UML metamodell kiterjesztésekhez;
- A négy szintű metamodell felépítményre alapozva architekturális alapot biztosít az UML metamodell más szabványokhoz való illesztéséhez.

A metamodellezéshez általánosan elfogadott fogalmi keret egy négyszintű architektúrán alapszik:

1. **meta-metamodell**
2. **metamodell**
3. **modell**
4. **felhasználói objektumok**

E szintek funkcióit a következő táblázat foglalja össze:

Szint	Leírás	Példa
meta-metamodell	Ez az infrastruktúra a metamodellezési architektúrához. Nyelvezetet definiál a metamodellek meghatározásához.	*MetaClass, MetaAttribute, MetaOperation*
metamodell	A meta-metamodell egy példánya. Nyelvezetet definiál a modellek meghatározásához.	*Class, Attribute, Operation, Component*
modell	A metamodell egy példánya. Nyelvezetet definiál egy információ terület leírásához.	*StockShare, askPrice, sellLimitOrder, StockQuoteServer*
felhasználói objektumok (felhasználói adat)	A modell egy példánya. Meghatározott információ területet definiál.	*<Acme_Software_Share_98789>, 654.56, sell_limit_order, <Stock_Quote_Svr_32123>*

A *meta-metamodellezési* szint képezi az alapot a metamodellezési architektúra számára. E szint elsődleges felelőssége nyelvezetet definiálni a metamodell meghatározásához. A meta-metamodell a metamodellnél magasabb absztrakciós szinten definiál egy modellt és jellegzetesen tömörebb mint az a metamodell, melyet leír. Egy meta-metamodell többszörös metamodelleket is definiálhat, és többszörös meta-metamodellek kapcsolatban állhatnak mindegyik metamodellel.[6] Általánosan elvárható ugyan, hogy az összefüggésben álló metamodellek és meta-metamodellek közös tervezési filozófiát és szerkezeteket kövessenek, de ez nem egy szigorú szabály. Minden egyes szint megkívánja a saját tervezési integritásának fenntartását. Példák a meta-metaobjektumokra a meta-metamodellezési szinten belül: *MetaClass, MetaAttribute, MetaOperation.*

Egy *metamodell* a meta-metamodell egy példánya. A metamodell szint elsődleges felelőssége nyelvezetet definiálni a modellek specifikációjához. A metamodellek jellegzetesen kidolgozottabbak mint a meta-metamodellek, melyek leírják őket, különösen akkor, amikor dinamikus szemantikát definiálnak.
Példák a metaobjektumokra a metamodellezési szinten belül: *Class, Attribute, Operation, Component Class, Attribute, Operation, Component*.

A *modell* a metamodell egy példánya. A modell szint elsődleges felelőssége nyelvezetet definiálni egy információ terület leírásához. Példák az objektumokra a modellezési szinten belül: *StockShare, askPrice, sellLimitOrder, StockQuoteServer*

A *felhasználói objektumok* (felhasználói adat) egy modell példányai. A felhasználói objektumok szint elsődleges felelőssége leírni egy meghatározott információterületet. Példák

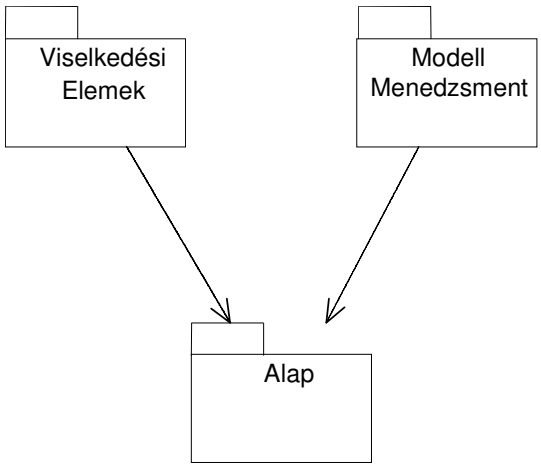
[6] Ha nem létezik egy kifejezett meta-metamodell, akkor egy implicit meta-metamodell tart fennt kapcsolatot minden metamodellel.

az objektumokra a felhasználói objektumok szinten belül: <*Acme_Software_Share_98789*>, *654.56, sell_limit_order,* <*Stock_Quote_Svr_32123*>

Az UML metamodellt úgy építették fel, hogy az az OMG Meta Objektum Szolgáltatások (Meta Object Facility) meta-metamodellből példányosítható. Az UML metamodell és a MOF meta-metamodell kapcsolata az *UML Proposal Summary* c. dokumentum függelékében megtalálható.

.3 Csomag struktúra

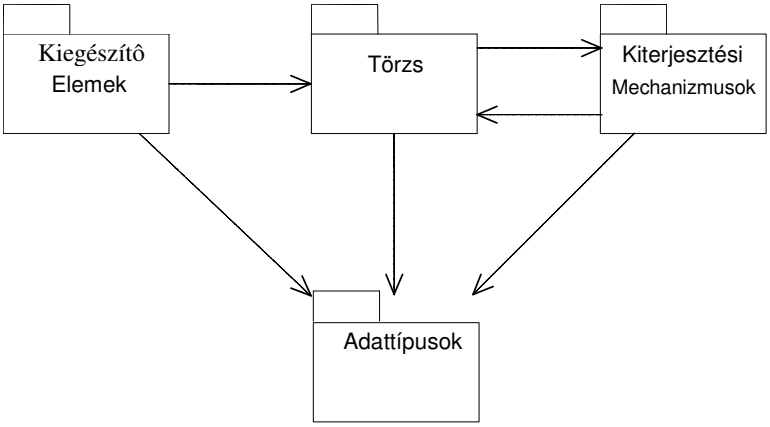
Az UML metamodell meglehetősen összetett. Körülbelül 90 metaosztályból, több mint 100 metaasszociációból és közel 50 sztereotípusból[7] (sztereotípiából) áll. Ennek az igen összetett metamodellnek a menedzselése logikai csomagokba történő szervezéssel történik. E csomagok csoportosítják a metaosztályokat, melyek erős kohéziót mutatnak egymással és laza kapcsolattal rendelkeznek más csomagbeli metaosztályokkal. Az UML metamodell legfelső szintű csomagokra való felbontása az 1. ábrán látható:



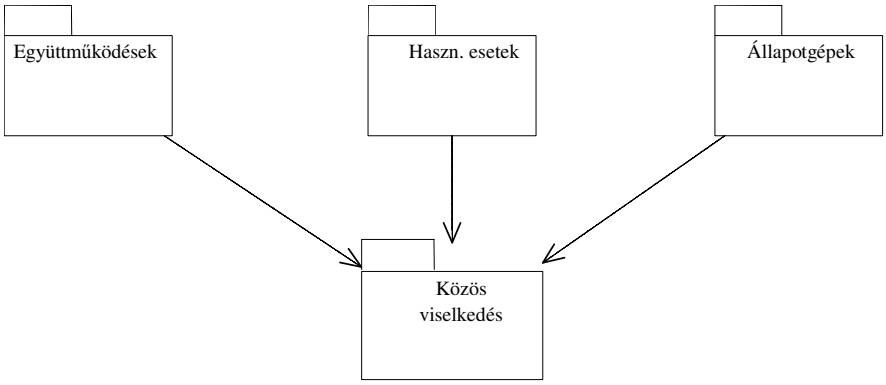
1. ábra: A legfelső szintű csomagok

Az Alap és a Viselkedési elemek csomagok további felbontása a 2. és 3. ábrán látható:

[7] Az Irodalomjegyzékben hivatkozott [2] sz. magyar nyelvű mű az angol “stereotype” kifejezést - feltehetően szakmai megfontolások alapján - “sztereotípus” nak fordítja, e hivatkozott mű “úttörő” jellegére és elterjedtségére való tekintettel a szakdolgozat is e kifejezést veszi át a fogalom lefedésére.



2. ábra: alap csomag



3. ábra: Viselkedési Elemek Csomag

E csomagok funkcióit és tartalmukat a dolgozat 2 - 4. része tárgyalja.

2 A nyelv formalizmusa

E szakasz az UML leírásához használt technikákat ismerteti. A specifikáció - a könnyű érthetőség fenntartása mellett - formális technikákat alkalmaz a precizitás növelése érdekében. A technika három nézőpontból, szövegesen és grafikusan egyaránt bemutatja az UML metamodellt. Az alkalmazott formális technikák előnyei a következők:

- javítja a leírás szabatosságát,
- csökkenti a félreérthetőséget és az inkonzisztenciát,
- kiegészítő technika által megerősíti a metamodell architektúráját,
- növeli a leírás érthetőségét.

Fontos megjegyezni, hogy e leírás nem teljes formális specifikációja a nyelvnek, mert jelentős mértékben kellene növelni a komplexitását a várható előnyökhöz képest.

A nyelv struktúrája mindemellett egy pontos specifikációt ad, mely elengedhetetlen az eszközök működőképességéhez. A dinamikus szemantikát természetes nyelv használatával írták le, de igen szabatos módon, így azok könnyen értelmezhetőek. Jelenleg a dinamikus szemantikát nem tekintjük nélkülözhetetlennek az eszközök fejlesztéséhez. Ez azonban valószínűleg változni fog a jövőben.

.1 A formalizmus szintjei

A nyelvek specifikációjának általános technikája, hogy először a nyelv *szintaxisát* definiáljuk, majd leírjuk a *statikus* és *dinamikus szemantikáját*. A szintaxis azt definiálja, hogy milyen szerkezetek vannak a nyelvben, és azok hogyan épülnek fel más konstrukciók szempontjából. Néha, különösen, ha a nyelvnek grafikus szintaxisa van, fontos más módon is, jegyzet formájában definiálni a szintaxist (pl. a nyelv absztrakt szintaxisának definiálása). A konkrét szintaxis ezután a jegyzet absztrakt szintaxissá való átdolgozásával definiálható. A szintaxis leírása az *Absztrakt szintaxis* fejezetben megtalálható.

Egy nyelv statikus szemantikái azt definiálják, hogy a szerkezet egy példánya hogyan kapcsolódjon más példányokhoz. A dinamikus szemantikák egy jól kialakított szerkezet jelentését definiálják. A leírás jelentése csak akkor definiált a nyelvben, ha a leírás jól kialakított (pl. ha eleget tesz a statikus szemantikában definiált szabályoknak). A statikus szemantika témakört a *Jól képzett szabályok* szakasz taglalja. A dinamikus szemantikáról a *Szemantika* szakaszban olvashatunk részletesebben. Néhány esetben a statikus szemantikák egyes részeit - a teljesség kedvéért - a *Szemantika* szakasz is részletezi.

A specifikáció különböző nyelvek kombinációját - az UML egy részhalmazát, egy objektum megszorítás nyelvet, és precíz természetes nyelvet - alkalmaz a teljes UML absztrakt szintaxisának és szemantikájának leírásához. A leírás nem hivatkozik más dokumentumokra,

megértéséhez nincs szükség egyéb információforrásra[8]. Az UML metamodell megalkotásakor különböző technikákat alkalmaztak az UML képességeinek felhasználásával. A fő nyelvi struktúrák metaosztályokként testesülnek meg a metamodellben. Más konstrukciók lényegében egyéb konstrukciók variánsai, és a metamodellbeli metaosztályok sztereotípiáikként definiálták őket. Ez a mechanizmus lehetővé teszi, hogy a konstrukció változatok szemantikái jelentősen eltérjenek az alap metaosztályokétól. Egy másik, még egyszerűbb módja e variánsok definiálásának a metaattributumok használata. Jó példa erre, hogy az *aggregáció* szerkezet a metaosztály egy attributumával határozták meg (*AssociationEnd*), melyet akkor alkalmazunk, ha jelezni kívánjuk, hogy egy asszociációs kapcsolat egy közönséges aggregát, egy kompozit aggregát, vagy egy általános asszociációs kapcsolat.

.2 Csomag specifikációs struktúra

A dolgozat az UML metamodell minden egyes csomagjával külön szakaszban foglalkozik. E szakaszok a következő alszakaszokat tartalmazzák:

- **Absztrakt Szintaxis** Az absztrakt szintaxist egy olyan diagramban mutatjuk be, mely ábrázolja a metaosztályokat, definiálja a konstrukciójukat és kapcsolataikat. A diagram ezen kívül még a jól képzett szabályok közül mutat be néhányat, főként a kapcsolatok multiplicitás szükségletét, valamint azt, hogy szükséges -e bizonyos alkonstrukció példányok rendezettsége. Végül minden konstrukciót el kell látni egy természetes nyelvű informális leírással. Minden leírás első paragrafusa azon konstrukciónak az általános bemutatása, mely a kontextust állítja be, míg a következő paragrafus az UML beli konstrukciót specifikáló metaosztály definíciója. Minden metaosztályhoz felsoroljuk attributumaikat azok rövid leírásaival. Mindezen kívül az asszociációs kapcsolatok ellentétes szerep neveit, (melyek a metaosztályokhoz kapcsolódnak), szintén hasonló módon soroljuk fel.

- **Jól képzett szabályok** Minden UML beli konstrukció statikus szemantikáját - kivéve a multiplicitást és a megszorítások elrendezését - a metaosztályokból létrehozott példányok invariánsainak halmazaként definiálták. Ezen invariánsoknak kielégítőeknek kell lenniül a megfelelő struktúrákhoz. A szabályok ily módon meghatározzák a metamodellben definiált attributumokra és asszociációs kapcsolatokra vonatkozó megszorításokat. Minden invariánst egy - egy OCL kifejezéssel definiáltak, és minden kifejezést infromális magyarázattal láttak el. Számos esetben további, a metaosztályokon elvégzett műveletek szükségesek az OCL kifejezésehkez. E műveleteket külön alfejezetben kell definiálni a konstrukcióhoz képzett szabályok után, és ugyanazt a megközelítést kell alkalmazni, mint az absztrakt szintaxisnál: informális magyarázat, melyet az operációt definiáló OCL kifejezés követ.

- **Szemantika** A konstrukciók jelentéseit természetes nyelv használatával definiálták. A konstrukciók logikai egységekbe vannak csoportosítva, melyeket együtt definiáltak. Mivel csak konkrét metaosztályoknak van igazi jelntésük egy nyelvben, e szakasz csak ezeket ismerteti.

[8] Az UML négyszintű metamodell felépítésének és a meta-metamodell alapjainak megértése igen nagy segítséget jelent a nyelv használatának elsajátításában, de nem előfeltétele az UML -szemantika értelmezéséhez.

- **Szabványos Elemek** Az előzőleg definiált metaosztályok sztereotípusai informális definícióval, természetes nyelven kifejezve.

- **Megjegyzés** Ez a rész természetes nyelven leírt magyarázatokat tartalmazhat a metamodellezési döntésekhez, pragmatikát a konstrukciók és példák használatához.

.3 A megszorítások nyelvezetének (OCL) használata

A következő konvenciókat használjuk az érthetőség elősegítése érdekében:

- "önmaga" (self) mely egy utalás a metaosztályra, definiálja az invariáns konkextust.

- Olyan kifejezésben, ahol egy csoport ismétlődik, egy iterátort használunk az érthetőség kedvéért, még akkor is, ha ez formálisan szükségtelen lenne. Az iterátor típusa általában elhagyható, de alkalmazni kell, ha az érthetőség ezt megkívánja.

- A "collect" operáció implicit módon alkalmazandó, ahol ez célszerű.

.4 A természetes nyelv használata

Mindenki számára nyilvánvaló törekvés a természetes nyelv pontos használata. Például az UML szemantikájának leírása tartalmaz olyan szerkezeteket, mint "X képességet biztosít ... számára ..." vagy "X egy Y". Ezen esetek mindegyikében a szokásos nyelv jelentése a feltételezett, jóllehet, egy erőteljesen formai leírás megköveteli a szemantika specifikációját még ezen egyszerű kifejezések esetében is.

A következő általálnos szabályokat alkalmazzuk:

Amikor valamely metaosztály egy példányára utalunk, gyakran elhagyjuk a "példány" szót. Például az "Osztály példány" vagy az "Asszociációs kapcsolat példány" helyett csak azt mondjuk, hogy "egy Osztály" vagy "egy Asszociáció". Az "egy" prefixum által azt feltételezük, hogy "valaminek egy példányára" gondolunk. Ugyanúgy, az "Elemek" szó használatával a "metaosztály Elemek példányainak egy halmazára" utalunk.

Bármikor, ha egy szó egybevág valamely UML -ben használt szerkezettel, akkor ez a szó e szerkezetre utal.

Az olyan kifejezések, melyek tartalmazzák az "al", "szuper", vagy "meta" prefixumok egyikét, egy szóként írandók (pl. metamodell, alosztály).

.5 Névkonvenciók és tipográfia

Az UML leírásában a következő konvenciókat használták:

Amikor UML -beli konstrukciókra utalunk és nem a metamodellbeli reprezentációikra, akkor normál szöveget alkalmazunk.

Amikor egy UML szerkezetet vagy mechanizmust említünk egy szövegkörnyezetben, akkor *dőlt bevűvel* írjuk.

A metaosztály neveket mindig nagybetűvel kezdjük és kisbetűkkel folytatjuk, ***dőlt és félkövér*** bevűket használva. Pl.: "Oszályozó". Az olyan nevekhez, melyek csatolt főnevet/melléknevet tartalmaznak, a szó közben nagy kezdőbetűket alkalmazunk (pl.: "ModellElem", "StruktúrálisTulajdonság").

A metaasszociáció és asszociáció osztályok neveit ugyanúgy írjuk, mint a metaosztályokét (pl.: "ElemReferencia").

A kis dőltbetűket az asszociáció végpontokhoz (szerep nevek), attrubutumokhoz és a metamodellbeli operációkhoz használjuk, pl. "*típus", "Tulajdonság"*. A szóközbeni nagy kezdőbetűket olyan nevekhez használjuk, melyek csatolt főnevet/melléknevet tartalmaznak.

A logikai metaattributum nevek mindig "is" -el (angol) kezdődnek (pl. "isAbstract").

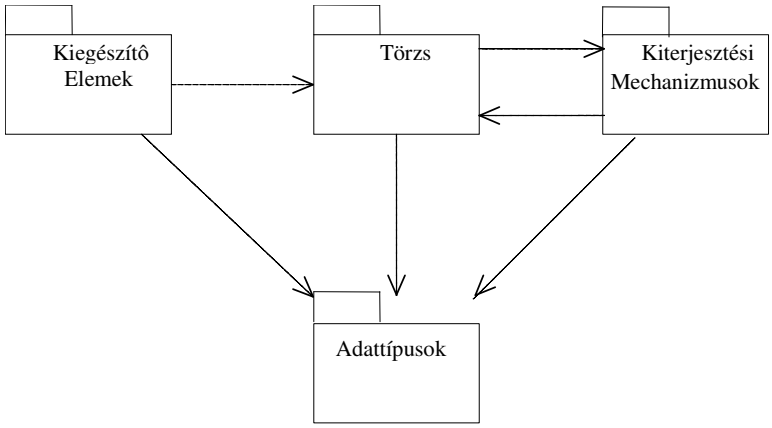
Amíg metaosztályokra, metaasszociációkra, metaattributumokra, stb. utalunk a szövegben, a nevekeiket pontosan ugyanúgy kell használni, mint ahogyan azok a modellben szerepelnek.

Az absztrakt metaosztályok neveit a metamodellben dőltbetűvel kell írni, de nem megkülönböztetve a szövegben szereplő más metaosztályoktól.

A sztereotípusok neveit kettős könyökzárójellel határoljuk el és kisbetűvel kezdjük (pl.: <<típus>>).

3. II. RÉSZ: ALAP CSOMAG

A második rész definiálja az infrastruktúrát az UML számára, ez az ún. Alap csomag. Az Alap csomag több alcsomagra bontható: Törzs, Kiegészítő elemek, Kiterjesztési mechanizmusok és Adattípusok. A Törzs csomag meghatározza azokat az alapfogalmakat, melyek egy elemi metamodellhez szükségesek, valamint definiál egy architekturális vázat a további nyelvi konstrukciók hozzáillesztéséhez, mint amilyenek a metaosztályok, metaasszociációk vagy a metaattributumok. A Kiegészítő elemek csomag további szerkezeteket definiál, melyek a Törzset terjesztik ki a további összetett fogalmak támogatására. Ilyenek pl. a függőségek, a sablonok, a fizikai struktúrák és a nézet elemek. A Kiterjesztési mechanizmusok csomag azt határozza meg, hogyan kell a modell elemeket testre szabni és új szemantikával kiterjeszteni. Az Adattípusok csomag alapvető adatstruktúrákat definiál a nyelv számára.



4. ábra: Alap csomagok

1 A nyelv törzse

.1 Áttekintés

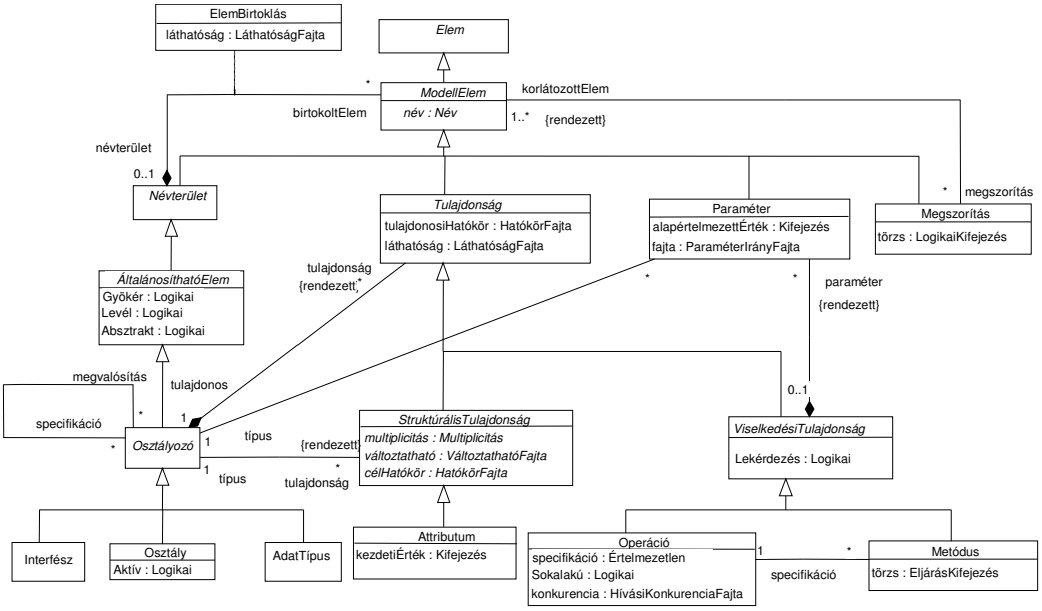
A *Törzs* csomag a legalapvetőbb azon alcsomagok közül, melyek az UML Alap csomagot alkotják. Ezen alcsomag definiálja az alap absztrakciót és konkrét szerkezeteket, melyek az objektum modellek fejlesztéséhez szükségesek. Az absztrakt metamodell szerkezetek nem példányosíthatók és általánosan használatosak a szerkezetek megtestesítéséhez, struktúrák megosztásához és a modell szervezéséhez. A konkrét metamodell szerkezetek példányosíthatók és az objektum modellezők (metamodellezők) által használt modellezési konstrukciókat tükrözik. A Törzs -ben definiált absztrakt szerkezetek magukba foglalják a ***ModellElem*** -et, az ***ÁltalánosíthatóElem*** **-**et és az ***Osztályozó*** -t. A konkrét szerkezetek, melyeket a Törzs -ben specifikáltak, magukba foglalják az ***Osztály*** -t, az ***Attributum*** -ot, az ***Operáció*** -t és az ***Asszociáció*** -t.

A Törzs csomag specifikálja az alapszerkezeteket, melyek egy alap metamodellhez szükségesek, valamint definiál egy architekturális vázat a további nyelvi konstrukciók hozzáillesztéséhez, mint amilyenek a metaosztályok, metaasszociációk vagy a metaattributumok. Igaz ugyan, hogy a Törzs csomag elegendő szemantikát tartalmaz az UML további részének definiálásához, de ez nem UML meta-metamodell. Ez az alapozás az Alap csomagon belül, mely infrastruktúraként szolgál a nyelv további részei számára. A többi csomagokban a Törzs kibővül a “csontváz” -hoz való metaosztályokkal, az általánosítások és asszociációk felhasználása mellett.

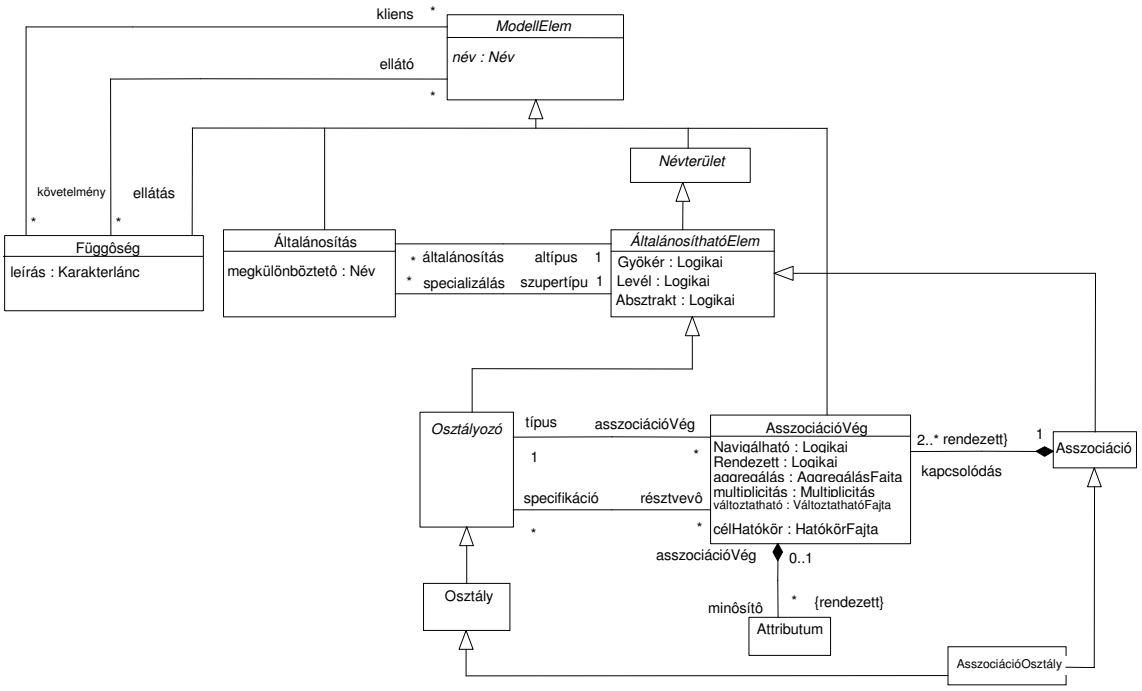
A következő szakaszok a Törzs csomagon belüli absztrakt szintaxist, a jól képzett szabályokat és a szemantikát írják le.

.2 Absztrakt szintaxis

A Törzs csomag absztrakt szintaxisa az 5. és 6. ábrán látható. Az 5. ábra a modell elemeket szemlélteti, melyek a metamodell struktúrális vázát alkotják. A 6. ábra azokat a modell elemeket mutatja be, melyek a kapcsolatokat definiálják.



5. ábra: Törzs csomag - Gerinc



6. ábra: Törzs csomag - Kapcsolatok

A Törzs csomag a következő metaosztályokat tartalmazza:

Asszociáció (Association)

Egy *asszociáció* szemantikus kapcsolatot definiál két osztályozó között; az asszociáció példányai (tuple) olyan halmazt alkotnak, melyek az osztályozók példányaival állnak kapcsolatban. E halmaz minden egyes elemének (tuple) értéke legfeljebb egyszer szerepelhet.

A metamodellben egy ***Asszociáció*** az ***Osztályozók*** (mint pl. egy ***Osztály***) közötti szemantikus kapcsolat deklarációja. Az **Asszociációnak** legalább két végpontja (AssociationEnds) van. Mindegyik végpont egy ***Osztályozóhoz*** kapcsolódik - ugyanaz az ***Osztályozó*** egynél több ***AsszociációVégponthoz*** is kapcsolódhat ugyanabban az ***Asszociációban***. Az ***Asszociáció*** az ***Osztályozó*** példányai közötti kapcsolatok halmazát reprezentálja. Az ***Asszociáció*** példánya egy ***Kapocs (Link)***, amelyik a megfelelő ***Osztályozóból*** létrehozott példányok összekötője.

Attributumok (Attributes)

név (name) *Az* **Asszociáció** *neve, kombinálva a hozzákapcsolt* **Osztályozóval**. Az őt körülvevő névterületen belül egyedinek kell lennie (általában egy ***Csomag [Package]***).

Asszociációk (Associations)

kapcsolódás (connection) Egy ***Asszociáció*** legalább két ***AsszociációVégpont*** ból áll, mindegyik végpont az asszociációnak az ***Osztályozóhoz*** való kapcsolódását reprezentálja. Minden ***AsszociációVégpont*** specifikál egy tulajdonság halmazt, melynek eleget kell tennie a kapcsolat

érvényességi feltételeinek. Egy ***Asszociáció*** struktúrájának definiálása az ***AsszociációVégpont*** által történik.

AsszociációOsztály (AssociationClass)

Egy *asszociáció osztály* olyan asszociáció, mely egyben osztály is. Nemcsak az osztályozók halmazát kapcsolja össze, hanem egy olyan tulajdonság halmazt is definiál, mely az önmagához való kapcsolódáshoz is tartozik, az osztályozók egyikéhez.

A metamodellben egy ***AsszociációOsztály*** olyan ***Osztályozó***k közötti kapcsolat deklarációja, melynek saját tulajdonság halmaza van. Az ***AsszociációOsztály*** az ***Asszociáció*** és az ***Osztály*** alosztálya. Ezért egy ***AsszociációOsztály***nak ***AsszociációVégpont***jai és ***Tulajdonság***ai is vannak.

AsszociációVégpont (AssociationEnd)

Egy ***AsszociációVégpont*** az asszociáció azon végpontja, amelyik az asszociációt az osztályozóhoz kapcsolja. Minden egyes ***AsszociációVégpont*** egy ***Asszociáció*** része; minden ***Asszociáció AsszociációVégpont***jai rendezettek.

A metamodellben egy ***AsszociációVégpont*** része egy ***Asszociáció***nak és ***Asszociációnak*** az***Osztályozóhoz*** való kapcsolódását specifikálja. Névvel rendelkezik, és a kapcsolat tulajdonságait definiálja.

A következőkben, amikor egy bináris asszociációbeli ***AsszociációVégpont***ra utalunk, a *forrás* végpont a másik végpont; a *cél* végpont az, amelyiknek a tulajdonságait tárgyaljuk.

Attributumok (Attributes)

aggregation Tartalmazási (egész-rész) kapcsolat. Amikor egy cél végpontot helyezünk el, meghatározzuk, hogy a cél végpont egy aggregáció, mely tekintettel van a forrás végpontra. Csak egy végpont lehet egy aggregáció. A lehetőségek:

- none — A végpont nem aggregátum.
- aggregate — A végpont aggregátum; a másik végpont ezért egy rész és az aggregáció értéke “none”. A részt tartalmazhatják más aggregátumok is.
- composite — A végpont egy kompozit (összetevő); a másik végpont ezért egy rész és az aggregáció értéke “none”. A részt erősen birtokolja a kompozit és nem lehet része más kompozitnak.

változtatható (changeable) Amikor egy cél végpontot helyezünk el, meghatározzuk, hogy az ***Asszociáció*** egy példánya módosítható -e a forrás végpontból. A lehetőségek:

- none — Nincs korlátozás a módostásra vonatkozólag.

frozen A forrás objektum létrehozása után nem helyezhetünk el kapcsokat (linkeket).

add only Bármikor elhelyezhetünk kapcsokat a forrásobjektumból, de a létrehozott kapocs nem távolítható el mielőtt legalább egy résztvevő objektumot nem szüntetünk meg.

rendezett (isOrdered) Amikor egy cél végpontot helyezünk el, meghatározzuk, hogy a forrás példányt és cél példányt összekötő kapcsok rendezettek -e. A rendezettséget azon ***Operáció***k által kell meghatározni és fenntartani, melyek a kapcsokat elhelyezik. A rendezettség olyan további információt reprezentál, mely az objektumokba vagy a kapcsokba az öröklési mechanizmuson keresztül nem kerül bele. A rendezett kapcsok halmaza sorrendben végigpáztázható. Létezik olyan alternatíva, hogy a kapcsoknak nincs hozzájuk tartozó rendezettségük.

navigálható (isNavigable) Amikor egy cél végpontot helyezünk el, meghatározzuk, hogy lehetséges -e keresztirányú kapcsolat a forrás példánytól a cél példányig. Minden egyes, az **Asszociáció**n keresztüli irány specifikációja független.

multiplicitás (multiplicity) Amikor egy cél végpontot helyezünk el, meghatározzuk azon cél példányok számát, melyek asszociációs kapcsolatban állhatnak egy szimpla forrás példánnyal a megadott **Asszociáció**n belül. (Ld. ***Multiplicitás***

név (name) A végpont szerep neve. A forrás osztályozó egy pszeudo-attributumát reprezentálja, ugyanolyan módon használható, mint egy ***Attributum*** , és egyedülállónak kell lennie az ***Attributum***okra és a forrás ***Osztályozó*** más pszeudo-attributumaira vonatkozólag.

cél hatáskör (targetScope) Meghatározza, hogy a célok közönséges ***Példányok***, vagy ***Osztályozó***k. A lehetséges esetek:

instance Az **Asszociáció** minden kapcsa tartalmaz egy hivatkozást a cél ***Osztályozó*** egy ***Példány***ára. Ez a környezet a normál ***Asszociációhoz***.

classifier Az ***Asszociáció*** minden kapcsa tartalmaz egy hivatkozást a cél ***Osztályozó***ra önmagára. Ez a lehetőség meta-információ tárolásra ad módot.

Asszociációk (Associations)

minősítő (qualifier) A minősítő ***Attributum***ok egy opcionális listája a végpont számára. Ha ez a lista üres, az ***Asszociáció*** nem minősített.

specifikáció (specification) Nulla, vagy több ***Osztályozót*** jelöl ki, melyek meghatározzák azon ***Operációkat***, melyek egy ***Példány***hoz alkalmazhatók és az ***AsszociációVégpont*** által az ***Asszociáció***n keresztül érhetők el. Ezek meghatározzák azt a minimális interfészt, mely az aktuális ***Osztályozó*** által realizálódik és az ***Asszociáció***s

kapcsolat érdekében kapcsolódik a végponthoz. Lehet egy ***Interfész***, vagy más ***Osztályozó.***

típus (type) Az ***Asszociáció*** végpontjához kapcsolt ***Osztályozó***t jelöli ki. Nem lehet ***Interfész***, mert nincs fizikai struktúrája.

Attributum (Attribute)

Az *attributum* egy névvel ellátott rész egy osztályozón belül, mely egy olyan értéktartományt ír le, mely értékek a tároló osztályozó értékei.

Attributumok (Attributes)

változtatható (changeable) Azt határozza meg, hogy az objektum előállítása után az adott érték módosítható -e. A lehetőségek:

none Nincs korlátozás a módosításra vonatkozólag.

frozen Az objektum példányosítása és értékeinek inicializálása után azok már nem módosíthatók és nem adhatók hozzá további értékek.

add only (Csak akkor van jelentősége, ha a multiplicitás nincs egy egyedi értékre rögzítve.) További értékek adhatók hozzá az értékhalmazhoz, de ha már létrejött egy érték, akkor az nem törölhető ill. módosítható.

kezdetiÉrték (initialValue) Olyan ***Kifejezés***, mely meghatározza az attributum inicializáláskori értékét, tehát az objektum inicializálásakor értékelődik ki. (Megjegyzendő, hogy egy külön konstruktor is előállíthat kezdeti értéket.)

multiplicitás (multiplicity) Az attributum számára fenntartott lehetséges adat értékek száma, melyeket egy példány fenntarthat.

Asszociációk (Associations)

típus (type) Kijelöli azt az osztályozót, melynek példányai az attributum értékei. ***Osztály***nak vagy ***Adattípusnak*** kell lennie.

ViselkedésiTulajdonság (BehavioralFeature)

A *viselkedési Tulajdonság* valamely modell elem egy dinamikus tulajdonságára utal, mint egy operáció vagy metódus.

A metamodellben a ***ViselkedésiTulajdonság*** valamely ***Osztályozó*** viselkedési aspektusát határozza meg. Az ***Osztályozó*** viselkedési aspektusainak összes különböző fajtája, amilyen az ***Operáció*** és a ***Metódus***, a ***ViselkedésiTulajdonság*** alosztályai. A ***ViselkedésiTulajdonság*** egy absztrakt metaosztály.

Attributumok (Attributes)

kérdés (isQuery) Meghatározza, hogy a ***Tulajdonság*** egy végrehajtása a rendszer állapotát változatlanul hagyja -e. Az "Igaz" (True) érték jelzi, hogy az

állapot változatlan, a "Hamis" érték azt fejezi ki, hogy mellékhatások fordulhatnak elő.

név (name) A ***Tulajdonság*** neve. A ***Tulajdonság*** egész szignatúrájának (név és paraméter lista) egyedinek kell lennie az őt tartalmazó ***Osztályozó***n belül.

Asszociációk (Associations)

paraméterek (parameters) Az ***Operáció Paraméter***einek rendezett listája. Az ***Operáció*** meghívásához a hívónak olyan értéklistát kell biztosítania, mely kompatibilis a ***Paraméterek*** típusával.

Osztály (Class)

Egy osztály olyan objektumok halmazának leírása, melyek attributumai, operációi, metódusai, kapcsolatai és szemantikája ugyanazok. Egy osztály interfészeket használhat a környezete számára biztosított műveletek meghatározására.

A metamodellben egy ***Osztály*** azon ***Objektum***okat írja le, melyek ugyanazokkal a ***Tulajdonság***okkal - ***Operáció***kkal, ***Attributum***okkal, ***Metódus***okkal - rendelkeznek. Ezen tulajdonságok az ***Objektum***ok halmazára nézve közösek. Ezen kívül egy ***Osztály*** nulla, vagy több ***Interfész***t is realizálhat; ez azt jelenti, hogy a *teljes leírójának* (ld. *Öröklődést [Inheritance]* a definícióhoz) tartalmaznia kell minden ***Operáció***t minden realizált ***Interfész***ből (tartalmazhat kiegészítő operációkat is).

Egy ***Osztály*** az ***Objektum***ok adatstruktúráit definiálja, néhány ***Osztály*** azonban lehet absztrakt, melyből közvetlenül nem lehet létrehozni ***Objektumokat***. Minden ***Objektum*** példányosítása egy olyan ***Osztály***ból történik, mely tartalmazza saját - a *teljes deszkriptor*ban deklarált ***ViselkedésiTulajdonság***nak megfelelő - értékhalmazát. Az ***Objektumok*** nem tartalmazhatnak a ***ViselkedésiTulajdonság***oknak vagy az osztály kiterjedés ***Attributum***ainak megfelelő értékeket. Egy ***Osztály*** minden ***Objektum***a az ***Osztály ViselkedésiTulajdonság***ainak definícióin osztozik és mindegyiküknek van hozzáférése azon egyedi értékhez, mely minden osztály kiterjedés attributum számára van fenntartva.

Attributumok (Attributes)

aktív (isActive) Meghatározza, hogy egy ***Objektum*** fenntart -e saját vezérlőszálat. Ha értéke "**Igaz**", akkor egy ***Objektumnak*** van saját vezérlőszála, mely konkurens módon fut más aktív ***Objektum***okkal. Ha az érték "**Hamis"**, akkor az ***Operációk*** a címterületben futnak, annak az ***Objektumnak*** a vezérlése alatt, mely a hívót vezérli.

Osztályozó (Classifier)

Az *osztályozó* olyan elem, mely viselkedési és struktúrális tulajdonságokat ír le; számos megjelenési formája van, mind pl. az osztály, az adattípus, az interfész és mások, melyek más metamodell csomagokban vannak definiálva.

A metamodellben egy ***Osztályozó Tulajdonságok*** csoportját deklarálja, mint pl. ***Attributumok, Metódusok*** és ***Operációk***. Névvel rendelkezik, mely egyedülálló az ***Osztályozó Névterületén*** belül. Az ***Osztályozó*** egy absztrakt metaosztály.

Asszociációk (Associations)

tulajdonság (feature) Az ***Osztályozó*** által birtokolt ***Tulajdonságok*** listája, ilyenek az ***Attributum, Operáció, Metódus***.

résztvevő (participant) A *specifikáció* inverze. Azt jelenti, hogy az ***Osztályozó*** részt vesz egy ***Asszociáció***ban.

megvalósítás (realization) ***Osztályozók*** halmaza, mely implementálja az ***Osztályozó Operációit***.

specifikáció (specification) ***Osztályozók*** azon halmaza, mely meghatározza azokat az ***Operációkat***, melyeket az ***Osztályozónak*** implementálnia kell. Az ***Osztályozó*** implementálhat több ***Operációt***, mint amennyit az ***Osztályozók*** halmaza tartalmaz. A halmaz tartalmazhat ***Interfészeket***, de ezek nem korlátozzák az elemeket.

Megszorítás (Constraint)

Egy *megszorítás* egy szemantikus feltétel vagy korlátozás.

A metamodellben egy ***Megszorítás*** egy - a kapcsolódó ***ModellElemek***en értelmezett - ***LogikaiKifejezés***, melynek igaz értékűnek kell lennie a modell helyes kialakításához. Ezt a korlátozást természetes, vagy jól definiált szemantikával rendelkező nyelven kell megadni. Bizonyos ***Megszorítások***at előre definiáltak az UML -ben (ld.: "B" melléklet), másokat a felhasználó definiálhat. Megjegyzendő, hogy egy ***Megszorítás*** egy kijelentés, nem pedig egy végrehajtható mechanizmus; egy olyan korlátozást jelöl, mely egy rendszer korrekt tervezése által jut érvényre.

Attributumok (Attributes)

body Egy ***LogikaiKifejezés,*** melynek igaznak kell lennie a kiértékeléskor a rendszer megfelelő kialakítása érdekében.

Asszociációk (Associations)

koráltozottElem (constrainedElement) ***ModellElem*** vagy ***ModellElemek*** listája, melyekre a ***Megszorítás*** kihat.

Adattípus (DataType)

Az *adattípus* olyan típus, mely értékeinek nincs azonosítója, így azok egyszerű értékek. Az adattípus fogalom magába foglalja a primitív beépített típusokat (mint pl. az egész és a karakterlánc), csakúgy mint a definiálható felsorolt típusokat (pl. a logikai típus egy olyan előre definiált felsorolt típus, melynek lehetséges értékei: "Igaz" vagy "Hamis").

A metamodellben egy ***AdatTípus*** a típus olyan speciális fajtáját definiálja, melyben az ***Operációk*** mind tiszta funkciók, így visszaadhatnak ***AdatÉrtékeket***, de nem tudják megváltoztatni az ***AdatÉrtékeket*** (mert nincsen azonosítójuk).

Függőség (Dependency)

Egy *függőség* megadja, hogy egy vagy több elem implementációjához vagy működéséhez egy vagy több más elem jelenléte szükséges. Minden elemnek léteznie kell a jelentés adott szintjén, azaz nem vonhatják maguk után az absztrakciós vagy realizációs szintek közötti váltást.

A metamodellben egy ***Függőség*** egy irányított kapcsolat egy kliens (vagy kliensek) -től egy ellátóig (vagy ellátókig), fenntartva, hogy a kliens függ az ellátótól, azaz a kliens elem megköveteli az ellátó elem jelenlétét és ismertségét.

A ***Függőségek*** sztereotípizálhatók a különböző fajta függőségek megkülönböztetésére.

Attributumok (Attributes)

leírás (description) A függőség szöveges leírása.

Asszociációk (Associations)

kliens (client) A ***ModellElem*** vagy ***ModellElem***ek halmaza, melyek megkövetelik az ellátó jelenlétét.

ellátó (supplier) A ***ModellElem*** vagy ***ModellElem***ek halmaza, melyek jelenlétét megköveteli a kliens.

Elem (Element)

Egy *elem* a modell egy atomi összetevője.

A metamodellen belül egy ***Elem*** a legfelső metaosztály a metaosztály hierarchiában. Két alosztálya van: ***ModellElem*** és ***NézetElem***. Az ***Elem*** egy absztrakt metaosztály.

ElemTulajdonviszony (ElementOwnership)

Az *elem tulajdonviszony* a névterületen belül látható.

A metamodellben az ***ElemTulajdonviszony*** (vagy ElemBirtoklás) a ***ModellElem*** és a ***Névterület*** között fennálló kapcsolatot testesíti meg és a ***ModellElem*** tulajdonosi viszonyát jelenti. Ld. ***ModellElem.***

Tulajdonság (Feature)

Egy *tulajdonság* olyan - az operációhoz, vagy az attributumhoz hasonló - jellemző, mely egy másik egyedben van egységbe zárva. Ilyen lehet egy interfész, egy osztály vagy egy adattípus.

A metamodellben egy ***Tulajdonság*** egy ***Osztályozó Példány*** vagy maga az ***Osztályozó*** viselkedési / struktúrális karakterisztikáját deklarálja. A ***Tulajdonság*** egy absztrakt metaosztály.

Attributes (Attributumok)

név (name) Az a név, mely a ***Tulajdonság***ot azonosítja az ***Osztályozón*** vagy a ***Példányon*** belül. Egyedinek kell lennie a nevek öröklődésén keresztül az ősöktől, beleértve a kimenő ***AsszociációVégpont***okat.

tulajdonosiHatáskör (ownerScope) A tulajdonosi viszony kiterjedési köre. Meghatározza, hogy a ***Tulajdonság*** az ***Osztályozó*** minden egyes ***Példányában*** megjelenik -e, vagy a ***Tulajdonság***nak csak egy egyedi példányában az egész ***Osztályozó***hoz. Az alábbi esetek lehetségesek:

instance Az ***Osztályozó*** minden egyes ***Példánya*** saját értékét tartja fenn a ***Tulajdonság*** számára.

classifier A ***Tulajdonságnak*** csak egyetlen értéke létezik az egész ***Osztályozóhoz.***

láthatóság (visibility) Meghatározza, hogy a ***Tulajdonság*** használható -e más ***Osztályozó*** által. A beágyazott ***Névterületek*** láthatóságai egyesülnek, így a legerőteljesebben korlátozó láthatóság az eredmény. A lehetőségek:

public Bármely külső ***Osztályozó***, amelyik láthatja az aktuális ***Osztályozót***, használhatja az adott ***Tulajdonságot***.

protected Az ***Osztályozó*** bármely leszármazottja használhatja a ***Tulajdonságot***.

private Csak az ***Osztályozó*** önmaga használhatja az adott ***Tulajdonságot.***

Asszociációk (Associations)

tulajdonos (owner) Az az ***Osztályozó***, mely tartalmazza a ***Tulajdonságot.***

ÁltalánosíthatóElem (GeneralizableElement)

Egy *általánosítható elem* olyan modell elem, mely részt vehet egy általánosítási kapcsolatban.

A metamodellben egy ***ÁltalánosíthatóElem*** lehet más ***ÁltalánosíthatóElem*** általánosítása, azaz minden ***Tulajdonság*** definiálva van az örökítőkben, melyek szintén jelen vannak az ***ÁltalánosíthatóElem***ekben, és az örökítők tartalmazzák a ***ModellElemeket***. Az ***ÁltalánosíthatóElem*** egy absztrakt metaosztály.

Attributes (Attributumok)

Absztrakt (isAbstract) Meghatározza, hogy az ***ÁltalánosíthatóElem*** egy befejezetlen deklaráció, vagy nem. Az "Igaz" érték jelzi, hogy az ***ÁltalánosíthatóElem*** egy befejezetlen deklaráció (absztrakt), a "Hamis" a teljes (konkrét)deklarációt jelöli. Egy absztrakt ***ÁltalánosíthatóElem*** nem példányosíthtó, mivel nem tartalmaz minden szükséges információt.

Levél (isLeaf) Meghatározza, hogy az aktuális ***ÁltalánosíthatóElem*** olyan ***ÁltalánosíthatóElem***, melynek nincs utóda (leszármazottja). Az "Igaz" érték jelzi, hogy az, és nem lehet hozzáadni utódokat, a "Hamis" érték jelöli, hogy hozzáadhatók utódok (függetlenül attól, hogy van -e jelenleg utóda).

Gyökér (isRoot) Meghatározza, hogy az aktuális ***ÁltalánosíthatóElem*** egy gyökér ***ÁltalánosíthatóElem*** ősök nélkül. "Igaz" érték jelzi, hogy az, és nem lehet hozzáadni ősöket; a "Hamis" érték jelöli, hogy hozzáadhatók ősök(függetlenül attól, hogy van -e jelenleg őse).

Asszociációk (Associations)

általánosítás (generalization) Kijelöl egy ***Általánosítást***, melynek **szupertípus** ***ÁltalánosíthatóEleme*** közvetlen őse a kurrens ***ÁltalánosíthatóElemnek.***

specializálás (specialization) Kijelöl egy ***Általánosítást***, melynek **altípus** ***ÁltalánosíthatóEleme*** közvetlen utóda a kurrens ***ÁltalánosíthatóElemnek.***

Általánosítás (Generalization)

Egy *általánosítás* osztályozástani kapcsolat egy általánosabb és egy speciálisabb elem között. A speciálisabb elem teljesen konzisztens az általánosabb elemmel (minden tulajdonsága, tagja és kapcsolata megvan); valamint további információt is tartalmazhat.

A metamodellben egy ***Általánosítás*** egy közvetlen örökítési kapcsolat, egyesítve egy ***ÁltalánosíthatóElem***et egy általánosabb ***ÁltalánosíthatóElemmel*** a hierarchián belül. Az ***Általánosítás*** egy altípus kapcsolat, azaz az általánosabb ***ÁltalánosíthatóElem Példánya*** helyettesíthető a specifikusabb ***ÁltalánosíthatóElem Példányával***. Ld. még: *öröklődést* az ***Általánosítási*** kapcsolat következményeihez.

Attributes (Attributumok)

diszkriminátor (discriminator) Megkülönböztető; kijelöl egy partíciót, melyhez az ***Általánosítás*** kapcsa tartozik. Az összes ***Általánosítási*** kapocs, mely egy megadott szupertípus ***ÁltalánosíthatóElem***en osztozik, csoportokra osztható a megkülönböztető nevük által. Minden kapocs - csoport egy megkülönböztető név alá tartozik. Ez a név a szupertípus ***ÁltalánosíthatóElem*** specializációjának ortogonális dimenzióját reprezentálja. A megkülönböztető névnek *nem* kell egyedinek lennie. Az üres karakterlánc egy újabb névként értelmeződik. Amennyiben az adott ***ÁltalánosíthatóElem*** összes ***Általánosítás***ának ugyanaz a neve (beleértve az üres nevet is), akkor ez az alárendelt elemek egyszerű halmaza. Egyébként az alárendelt elemek kettő, vagy több csoportot alkotnak, mindegyiket egy tagjuk, mint ős reprezentál egy konkrét utód elemben.

Asszociációk (Associations)

szupertípus (supertype) Kijelöl egy ***ÁltalánosíthatóElem***et, mely az altípus ***ÁltalánosíthatóElem*** általánosított verziója.

altípus (subtype) Kijelöl egy ***ÁltalánosíthatóElem***et, mely a szupertípus ***ÁltalánosíthatóElem*** specializált verziója.

Interfész (Interface)

Az *interfész* olyan operációk csoportjának deklarációja, melyeket a példány által felkínált szolgáltatások definíciójához használunk.

A metamodellben egy ***Interfész Operációk*** halmazát tartalmazza, melyek együtt definiálnak egy ***Osztályozó*** által felkínált szolgáltatást. Egy ***Osztályozó*** felkínálhat több szolgáltatást, ez azt jelenti, hogy több ***Interfészt*** realizálhat, ill. több ***Osztályozó*** is realizálhatja ugyanazt az ***Interfész***t.

Az ***Interfész***ek ***ÁltalánosíthatóElemek***. Minden ***Operáció*** deklarációja egy örökös által valósul meg, melynek vagy új ***Operáció***nak, vagy az ős(ök)ben deklarált ***Operáció***k specializációinak (korlátozásoknak) kell lennie.

Az ***Interfész***eknek nem lehetnek ***Attributumai, Asszociációi*** ill. ***Metódusai.***

Metódus (Method)

A *metódus* egy operáció implementációja. Specifikálja azt az algoritmust vagy eljárást, mely hatással van az operáció eredményeire.

A metamodellben egy ***Metódus*** egy ***Osztályozó***n belüli, névvel ellátott viselkedés deklarációja. A ***Metódus*** az ***Osztályozó*** egy vagy több ***Operáció***ját realizálja.

Attributes (Attributumok)

törzs (body) A ***Metódus*** implementációja ***ProceduralisKifejezés*** formájában.

Asszociációk (Associations)

specifikáció (specification) Kijelöl egy ***Operációt***, melyet a ***Metódus*** implementál. Az ***Operációt*** birtokolnia kell annak az ***Osztályozó***nak, mely a ***Metódus***t birtokolja, vagy abból kell örökíteni. Az ***Operáció*** és a ***Metódus*** jelölésének illeszkednie kell.

ModellElem (ModelElement)

Egy *modell elem* egy olyan elem, mely a modellezett rendszer egy absztrakciója, ellentétben a *nézet elem*mel, amelynek rendeltetése, hogy könnyen átlátható módon prezentáljon infromációt.

A metamodellben egy ***ModellElem*** egy névvel ellátott egyed a ***Modell***en belül. Ez képezi az alapot az összes modellezési metaosztály számára az UML -ben. Minden más modellezési metaosztály direkt vagy indirekt alosztálya a ***ModellElem***nek. A ***ModellElem*** egy absztrakt metaosztály.

Attributes (Attributumok)

név (name) Azonosító a ***ModellElem*** számára az őt tartalmazó ***Névterület***en belül.

Asszociációk (Associations)

megszorítás (constraint) Azon ***Megszorítás***ok halmaza, melyek az elemre hatással vannak.

ellátás (provision) Az ***ellátó*** inverze. Olyan ***Függőséget*** jelöl ki, melyben a ***ModellElem*** egy ellátó.

igény (requirement) A ***Kliens*** inverze. Olyan ***Függőséget*** jelöl ki, melyben a ***ModellElem*** egy kliens.

névterület (namespace) Kijelöli azon ***Névterületet***, mely a ***ModellElem***et tartalmazza. A gyökér elem kivételével minden ***ModellElem***nek pontosan egy ***Névterület***hez kell tartoznia. A ***Névterület*** nevek útvonalneve minden ***ModellElem*** számára egyedi kijelölést biztosít. Az asszociáció ***láthatóság*** attributuma meghatározza az elemnek a névterületen kívüli láthatóságát (ld. ***Láthatóság***).

Névterület (Namespace)

A *névterület* egy modell azon része, melyben minden névnek egyedi jelentésűnek kell lennie.

A metamodellben egy ***Névterület*** olyan ***ModellElem***, mely birtokolhat más ***ModellElem***eket, mint amilyenek az ***Asszociációk***, és az ***Osztályozók***. Minden birtokolt ***ModellElem*** nevének egyedinek kell lennie a ***Névterület***en belül. Továbbá minden tartalmazott ***ModellElem***et birtokolnia kell egy ***Névterületnek***. A ***Névterület*** konkrét alosztályának további megszorításai vannak, melyeket az elemek különböző fajtái tartalmazhatnak. A ***Névterület*** egy absztrakt metaosztály.

Asszociációk (Associations)

birtokoltElem (ownedElement) ***ModellElemek*** halmaza, melyeket a ***Névterület*** birtokol.

Operáció (Operation)

Egy *operáció* olyan szolgáltatás, melyet egy objektum kérhet. Az operáció egy szignatúrával rendelkezik, mely a lehetséges aktuális paramétereket írja le (beleértve a lehetséges visszatérési értékeket).

A metamodellben egy ***Operáció*** egy ***ViselkedésiTulajdonság***, mely az ***Osztályozó*** azon ***Példányai***hoz alkalmazható, melyek tartalmazzák az adott ***Operációt.***

Attributes (Attributumok)

konkurencia (concurrency) Meghatározza a konkurens hívások szemantikáját ugyanahhoz a passzív példányhoz, azaz egy ***Példány*** egy

Osztályozótól származik az *isActive=false* attributumértékkel. (Az aktív példányok felügyeletük alatt tartják a saját ***Operációik***hoz való hozzáférést, így ez a tulajdonságot általában [bár nem követelmény az UML -en belül] *sequential* értékre szokás állítani) A lehetőségek:

sequential Szekveciális. A hívókat úgy kell koordinálni, hogy egyidejűleg csak egy hívás menjen ki egy ***Példány***hoz (bármely szekvenciális ***Operáció***ra vonatkozólag). Ha ezen attrubutum érvényessége mellett szimultán hívások fordulnak elő, akkor a rendszer szemantikája és integritása nem garantálható.

guarded Védett. Egyidejűleg többszörös hívások is előfordulhatnak a konkurens szálakból egy ***Példány*** felé (bármely védett ***Operáció***ra), de csak az egyik számára engedélyezett a művelet megkezdése; a többiek blokkolt állapotban maradnak mindaddig, amíg az első ***Operáció*** be nem fejeződött. A rendszer tervező felelőssége, hogy ne fordulhasson elő kölcsönös kizárás a szimultán blokkolásokból kifolyólag. A védett ***Operációknak*** hibátlanul kell végrehajtódniuk (vagy blokkolniuk kell önmagukat), amennyiben szimultán szekvenciális ***Operációk***, vagy nem igényelhetnek védett szemantikát.

concurrent Konkurens. Egyidejűleg többszörös hívások is előfordulhatnak a konkurens szálakból egy ***Példány*** felé (bármely védett ***Operáció***ra). Mindegyikük végrehajtása konkurens módon történhet, korrekt szemantikával. A konkurens ***Operációknak*** hibátlanul kell végrehajtódniuk amennyiben szimultán szekvenciális vagy védett ***Operációk***, vagy nem igényelhetnek konkurens szemantikát.

sokalakú (isPolymorphic) Meghatározza, hogy az ***Operáció*** implementációját felülbírálhatja -e az alosztály. Ha az attributum értéke "Igaz", akkor az alosztályokon definiálhatók ***Metódusok***. Ha az attributum értéke "Hamis", akkor a ***Metódus*** megvalósítja az ***Operációt*** a kurrens ***Osztályozó***ban, mely változatlanul öröklődik az összes utód által.

specifikáció (specification) Az ***Operáció*** végrehajtásával járó hatások leírása ***Kifejezés*** formában.

Paraméter (Parameter)

A *paraméter* egy kötetlen változó, mely megváltoztatható, átadható, vagy visszaadható. Egy paraméter tartalmazhat egy nevet, típust, valamint a kommunikáció irányát. A paraméterek az operációk, üzenetek, események, sablonok, stb. specifikációjához használatosak.

A metamodellben egy ***Paraméter*** egy olyan argumentum deklarációja, melyet az ***Operáció***hoz***, Jel***hez***,*** stb. küldünk/fogadunk.

Attributes (Attributumok)

alapértelmezettÉrték (defaultValue) Alapértelmezett érték. ***Kifejezés***, melynek kiértékelése olyan értéket ad, melyet akkor használunk, amikor nincs biztosítva argumentum a ***Paraméter***hez.

fajta (kind) Meghatározza a kívánt ***Paraméter*** fajtáját. Lehetőségek:

in	Input ***Paraméter*** (nem módosítható)
out	Output ***Paraméter*** (módosítható a hívóval történő kommunikációhoz).
inout	Input ***Paraméter***, mely módosítható.
return	Egy hívás visszatérési értéke.

név (name) A ***Paraméter*** neve, melynek egyedinek kell lennie az őt tartalmazó ***Paraméter*** listán belül.

Asszociációk (Associations)

típus (type) Kijelöl egy ***Osztályozó***t, melyhez egy argumentum értéknek illeszkednie kell.

StruktúrálisTulajdonság (StructuralFeature)

A *struktúrális tulajdonság* egy modell elem olyan statikus jellemzőjére utal, amilyen egy attributum.

A metamodellben a ***StruktúrálisTulajdonság*** az ***Osztályozó Példány***ának egy struktúrális aspektusát deklarálja, mint amilyen egy ***Attributum***. Meghatározza például a ***StruktúrálisTulajdonság*** multiplicitását és változtathatóságát. A ***StruktúrálisTulajdonság*** egy absztrakt metaosztály.

Ld. ***Attributum***ot az attributumok és az asszociációk leírásához, mivel ez a ***StruktúrálisTulajdonság***nak csak alosztálya a kurrens metamodellen belül.

.3 Jól képzett szabályok

A következőkben ismertetendő jól képzett szabályokat alkalmazzuk a Törzs csomaghoz.

Asszociáció (Association)

[1] Az ***AsszociációVégpont***nak az ***Asszociáció***n belül egyedi nevének kell lennie.

```
self.allConnections->forAll( r1, r2 | r1.name = r2.name implies r1 = r2 )
```

[2] Az ***AsszociációVégpont*** leggyakrabban aggregáció vagy kompozíció.

```
self.allConnections->select(aggregation <> #none)->size <= 1
```

[3] Ha egy ***Asszociáció***nak 3 vagy több ***AsszociációVégpont***ja van, akkor az ***AsszociációVégpont*** lehet aggregáció vagy kompozíció.

[4] Az ***AsszociációVégpontok*** kapcsolódó ***Osztályozóinak*** az ***Asszociáció Névterületén*** belül kell lennie.

```
self.allConnections->forAll (r |
   self.namespace.allContents->includes (r.type) )
```

Kiegészítő Operációk

[1] Az *allConnections* operáció az ***Asszociáció*** összes ***AsszociációVégpont*** halmazát adja eredményül.

```
allConnections : Set(AssociationEnd);
allConnections = self.connection
```

AsszociációOsztály (AssociationClass)

[1] Az ***AsszociációVégpont***ok és a ***StruktúrálisTulajdonság***ok nevei nem lehetnek átfedésben.

```
self.allConnections->forAll( ar |
   self.allFeatures->forAll( f |
      f.oclIsKindOf(StructuralFeature) implies ar.name <> f.name ))
```

[2] Egy ***AsszociációOsztály*** nem definiálható önmaga és más között.

```
self.allConnections->forAll(ar | ar.type <> self)
```

Kiegészítő Operációk

[1] Az *allConnections* operáció az ***AsszociációOsztály*** összes ***AsszociációVégpont*** halmazát adja eredményül, beleértve az összes olyan kapcsolatot, melyet az ő szupertípusa (tranzitív bezárás) definiál.

```
allConnections : Set(AssociationEnd);
allConnections = self.connection->union(self.supertype->select
   (s | s.oclIsKindOf(Association))->collect (a : Association |
      a.allConnections))->asSet
```

AsszociációVégpont (AssociationEnd)

[1] Egy ***AsszociációVégpont Osztályozó***ja nem lehet ***Interfész*** vagy ***AdatTípus*** , kivéve, ha az ***AdatTípus*** része egy kompozit aggregációnak.

```
not self.type.oclIsKindOf (Interface)
and
( self.type.oclIsKindOf (DataType) implies
   self.association.connection->select ( ae | ae <> self)->forAll ( ae |
      ae.aggregation = #composite) )
```

[2] Egy ***Példány*** egy kompozíciónál nem tartozhat több, mint egy kompozit ***Példány***hoz.

```
self.aggregation = #composite implies self.multiplicity.max <= 1
```

Attributum (Attribute)

Nincsenek külön szabályok.

ViselkedésiTulajdonság (BehavioralFeature)

[1] Minden ***Paraméter***nek egyedi névvel kell rendelkeznie.

```
self.parameter->forAll(p1, p2 | p1.name = p2.name implies p1 = p2)
```

[2] A ***Paraméter***ek típusát tartalmaznia kell az ***Osztályozó Névterület***ének.

```
self.parameter->forAll( p |
   self.owner.namespace.allContents->includes (p.type) )
```

Kiegészítő Operációk

[1] A *hasSameSignature* operáció azt vizsgálja, hogy az argumentumnak ugyanaz -e a szignatúrája, mint magának a példánynak.

```
hasSameSignature ( b : BehavioralFeature ) : Boolean;
hasSameSignature (b) =
   (self.name = b.name) and
   (self.parameter->size = b.parameter->size) and
   Sequence{ 1..(self.parameter->size) }->forAll( index : Integer |
      b.parameter->at(index).type =
         self.parameter->at(index).type and
      b.parameter->at(index).kind =
         self.parameter->at(index).kind
   )
```

Osztály (Class)

[1] Amennyiben egy ***Osztály*** konkrét, az ***Osztály*** minden ***Operáció***jának kell lennie egy őt megvalósító ***Metódus***nak a *teljes deszkriptor*ban.

```
not self.isAbstract implies self.allOperations->forAll (op |
self.allMethods->exists (m | m.specification->includes(op)))
```

[2] Egy ***Osztály*** csak ***Osztályokat, Asszociáció***kat, ***Generalizáció***kat, ***HasználatiEset***eket, ***Megszorítás***okat, ***Függőség***eket, ***Együttműködés***eket és ***Interfész***eket tartalmazhat, mint ***Névterület***.

```
self.allContents->forAll->(c |
   c.oclIsKindOf(Class         ) or
   c.oclIsKindOf(Association   ) or
   c.oclIsKindOf(Generalization) or
   c.oclIsKindOf(UseCase       ) or
   c.oclIsKindOf(Constraint    ) or
   c.oclIsKindOf(Dependency    ) or
   c.oclIsKindOf(Collaboration ) or
   c.oclIsKindOf(Interface     )
)
```

[3] Egy ***Interfész***ben minden egyes ***Operáció*** számára az ***Osztály*** által biztosított, hogy az ***Osztály***nak lennie kell egy illeszkedő ***Operáció***jának.

```
self.specification.allOperations->forAll (interOp |
   self.allOperations->exists( op | op.hasSameSignature (interOp) ) )
```

Osztályozó (Classifier)

[1] Azonos fajta ViselkedésiTulajdonságoknak nem lehet ugyanaz a szignatúrája az ***Osztályozó***n belül.

```
self.feature->forAll(f, g |
   (
      (
         (f.oclIsKindOf(Operation) and g.oclIsKindOf(Operation)) or
         (f.oclIsKindOf(Method   ) and g.oclIsKindOf(Method   )) or
         (f.oclIsKindOf(Reception) and g.oclIsKindOf(Reception))
      ) and
      f.oclAsType(BehavioralFeature).hasSameSignature(g)
   )
implies f = g)
```

[2] Az ***Attributum***oknak nem lehet ugyanaz a nevük az ***Osztályozó***n belül.

```
self.feature->select ( a | a.oclIsKindOf (Attribute) )->forAll ( p, q |
   p.name = q.name implies p = q )
```

[3] Az oppozit (ellenoldali) ***AsszociációVégpont***oknak nem lehet ugyanaz a nevük az ***Osztályozó***n belül.

```
self.oppositeEnds->forAll ( p, q | p.name = q.name implies p = q )
```

[4] Az ***Attributum*** neve nem lehet ugyanaz, mint egy ellenoldali ***AsszociációVégpont*** vagy egy - az ***Osztályozó*** által tartalmazott - ***ModellElem*** neve.

```
self.feature->select ( a | a.oclIsKindOf (Attribute) )->forAll ( a |
   not self.allOppositeAssociationEnds->union (self.allContents)->collect ( q |
q.name )->includes (a.name) )
```

[5] Egy ellenoldali ***AsszociációVégpont*** neve nem lehet ugyanaz, mint egy ***Attributum*** vagy egy - az ***Osztályozó*** által tartalmazott - ***ModellElem*** neve.

```
self.oppositeAssociationEnds->forAll ( o |
  not self.allAttributes->union (self.allContents)->collect ( q |
    q.name )->includes (o.name) )
```

Kiegészítő Operációk

[1] Az *allFeatures* operáció egy olyan halmazt ad eredményül, mely az ***Osztályozó*** összes saját és örökölt ***Tulajdonság***át tartalmazza.

```
allFeatures : Set(Feature);
allFeatures = self.feature->union(
                      self.supertype.oclAsType(Classifier).allFeatures)
```

[2] Az *allOperations* operáció egy olyan halmazt ad eredményül, mely az ***Osztályozó*** összes saját és örökölt ***Operáció***ját tartalmazza.

```
allOperations : Set(Operation);
allOperations = self.allFeatures->select(f | f.oclIsKindOf(Operation))
```

[3] Az *allMethods* operáció egy olyan halmazt ad eredményül, mely az ***Osztályozó*** összes saját és örökölt ***Metódus***át tartalmazza.

```
allMethods : set(Method);
allMethods = self.allFeatures->select(f | f.oclIsKindOf(Method))
```

[4] Az *allAttributes* operáció egy olyan halmazt ad eredményül, mely az ***Osztályozó*** összes saját és örökölt ***Attributumát*** tartalmazza.

```
allAttributes : set(Attribute);
allAttributes = self.allFeatures->select(f | f.oclIsKindOf(Attribute))
```

[5] Az *associations* operáció egy olyan halmazt ad eredményül, mely az ***Osztályozó*** összes saját ***Asszociáció***ját tartalmazza.

```
associations : set(Association);
associations = self.associationEnd.association->asSet
```

[6] Az *allAssociations* operáció egy olyan halmazt ad eredményül, mely az ***Osztályozó*** összes saját és örökölt ***Asszociáció***ját tartalmazza.

```
allAssociations : set(Association);
allAssociations = self.associations->union (
                  self.supertype.oclAsType(Classifier).allAssociations)
```

[7] Az *oppositeAssociationEnds* operáció egy olyan halmazt ad eredményül, mely az ***Osztályozó***ra nézve oppozit (ellentétes) összes ***AsszociációVégpont***ot tartalmazza.

```
oppositeAssociationEnds : Set (AssociationEnd);
oppositeAssociationEnds =
   self.association->select ( a | a.associationEnd->select ( ae |
      ae.type = self ).size = 1 )->collect ( a |
         a.associationEnd->select ( ae | ae.type <> self ) )->union (
   self.association->select ( a | a.associationEnd->select ( ae |
      ae.type = self ).size > 1 )->collect ( a |
         a.associationEnd) )
```

[8] Az *AllOppositeAssociationEnds* operáció egy olyan halmazt ad eredményül, mely az ***Osztályozó***ra nézve oppozit (ellenoldali) összes ***AsszociációVégpont***ot tartalmazza, beleértve az örökölteket is.

```
allOppositeAssociationEnds : Set (AssociationEnd);
allOppositeAssociationEnds = self.oppositeAssociationEnds->union (
      self.supertype.allOppositeAssociationEnds )
```

Megszorítás (Constraint)

[8] ***Megszorítás*** nem alkalmazható sajátmagához.

```
not self.constrainedElement->includes (self)
```

AdatTípus (DataType)

[1] Egy ***AdatTípus*** csakis lekérdező jellegű ***Operáció***kat tartalmazhat.

```
self.allFeatures->forAll(f |
               f.oclIsKindOf(Operation) and f.oclAsType(Operation).isQuery)
```

[2] Egy ***AdatTípus*** nem tartalmazhat bármilyen más ***ModellElem***et.

```
self.allContents->isEmpty
```

Függőség (Dependency)

Nincsenek külön szabályok.

Elem (Element)

Nincsenek külön szabályok.

ElemTulajdonosiViszony (ElementOwnership)

Nincsenek kiegészítő szabályok.

ÁltalánosíthatóElem (GeneralizableElement)

[1] Egy gyökérnek semmilyen ***Általánosítás***a.

```
self.isRoot implies self.generalization->isEmpty
```

[2] Egy ***ÁltalánosíthatóElem***nek nem lehet olyan szupertípus ***Általánosítás***a, amely levél elemre vonatkozik.

```
self.supertype->forAll(s | not s.isLeaf)
```

[3] A cirkuláris öröklődés nem engedélyezett.

```
not self.allSupertypes->includes(self)
```

[4] Az ***ÁltalánosíthatóElem Névterület***ének tartalmaznia kell a szupertípust.

```
self.generalization->forAll(g |
```

```
    self.namespace.allContents->includes(g.supertype) )
```

Kiegészítő Operációk

[1] Az *allContents* operáció egy olyan halmazt ad eredményül, mely az ***ÁltalánosíthatóElem***ben tárolt összes ***ModellElem*** tartalmazza a szupertípustól örökölt tartalommal együtt.

```
allContents : Set(ModelElement);
allContents = self.contents->union(
  self.supertype.allContents->select(e |
     e.elementOwnership.visibility = #public or
     e.elementOwnership.visibility = #protected))
```

[2] A *supertype* operáció egy olyan halmazt ad eredményül, mely az összes közvetlen szupertípust tartalmazza.

```
supertype : Set(GeneralizableElement);
supertype = self.generalization.supertype
```

[3] Az *AllSupertype* operáció egy olyan halmazt ad eredményül, mely az adott ***ÁltalánosíthatóElem***ből örökölt összes ***ÁltalánosíthatóElem***et tartalmazza (tranzitív bezárás), kivéve magát az adott ***ÁltalánosíthatóElem***et.

```
allSupertypes : Set(GeneralizableElement);
allSupertypes = self.supertype->union(self.supertype.allSupertypes)
```

Általánosítás (Generalization)

[1] Egy ***ÁltalánosíthatóElem***nek csak egy ugyanolyan fajtájú ***ÁltalánosíthatóElem*** alosztálya lehet.

```
self.subtype.oclType = self.supertype.oclType
```

Interfész (Interface)

[1] Egy ***Interfész*** csak ***Operáció***kat tartalmazhat.

```
self.allFeatures->forAll(f | f.oclIsKindOf(Operation))
```

[2] Egy ***Interfész*** semmilyen ***Osztályozó***t nem tartalmazhat.

```
self.allContents->isEmpty
```

[3] Az összes ***Tulajdonság*** publikus, amely egy ***Interfész***ben van definiálva.

```
self.allFeatures->forAll ( f | f.visibility = #public )
```

Metódus (Method)

[1] Ha egy megvalósított ***Operáció*** lekérdezés, akkor az ***Metódus*** is lehet.

```
self.specification->exists ( op | op.isQuery ) implies self.isQuery
```

[2] A ***Metódus*** szignatúrájának ugyanolyannak kell lennie, mint a megvalósított ***Operáció***é.

```
self. specification->forAll ( op | self.hasSameSignature (op) )
```

[3] A ***Metódus*** láthatóságának ugyanolyannak kell lennie, mint a megvalósított ***Operáció***é.

```
self. specification->forAll ( op | self.visibility = op.visibility )
```

ModellElem (ModelElement)

Kiegészítő Operációk

[1] A *supplier (ellátó)* operáció eredményei azon halmaz elemei, mely a ***ModellElem*** összes közvetlen ellátóját tartalmazza.

```
supplier : Set(ModelElement);
supplier = self.provision.supplier
```

[2] Az *allSupplier (összes ellátó)* operáció eredményei azon halmaz elemei, mely tartalmazza mindazon ***ModellElem***et, melyek az aktuális ***ModellElem*** ellátói , beleértve. Ez egy tranzitív bezárás.

```
allSuppliers : Set(ModelElement);
allSuppliers = self.supplier->union(self.supplier.allSuppliers)
```

[3] A *model* operáció eredményei azon ***Modell*** egységei, melyek egy ***ModellElem***hez tartoznak.

```
model : Set(Model);
model = self.namespace->union(self.namespace.allSurroundingNamespaces)
         ->select( ns|
               ns.oclIsKindOf (Model))
```

Névterület (Namespace)

[1] Ha egy tartalmazott elemnek, mely nem ***Asszociáció*** vagy ***Általánosítás***, van neve, akkor ennek a névnek egyedinek kell lennie a ***Névterület***en belül.

```
self.allContents->forAll(me1, me2 : ModelElement |
   (  not me1.oclIsKindOf (Association) and not me2.oclIsKindOf (Association) and
      me1.name <> '' and me2.name <> '' and me1.name = me2.name
   ) implies
      me1 = me2 )
```

[2] Minden ***Asszociáció***nak kell rendelkeznie egy egyedi névkombinációval és kapcsolódó ***Osztályozó***val a ***Névterület***en belül.

```
self.allContents->select(oclIsKindOf(Association))->
   forAll(a1, a2 : Association |
      (  a1.name = a2.name and
         a1.connection->size = a2.connection->size and
         Sequence{1..a1.connection->size}->forAll(i |
            a1.connection->at(i).type = a2.connection->at(i).type)
      ) implies
         a1 = a2)
```

Kiegészítő Operációk

[1] A *contents* operáció eredményei azon halmaz elemei, mely az összes olyan ***ModellElem***et magába foglalja, melyet a ***Névterület*** tartalmaz.

```
contents : Set(ModelElement)
contents = self.ownedElement
```

[2] Az *allContents* operáció eredményei azon halmaz elemei, mely az összes olyan ***ModellElem***et magába foglalja, melyet a ***Névterület*** tartalmaz.

```
allContents : Set(ModelElement);
allContents = self.contents
```

[3] Az *allVisibleElements* operáció eredményei azon halmaz elemei, mely magába foglalja az összes olyan ***ModellElem***et, mely a ***Névterület***en kívülről látható.

```
allVisibleElements : Set(ModelElement)
allVisibleElements = self.allContents->select(e |
   e.elementOwnership.visibility = #public)
```

[4] Az *allSurroundingNamespaces* operáció eredményei azon halmaz elemei, mely magába foglalja az összes környező ***Névterület***et.

```
allSurroundingNamespaces : Set(Namespace)
allSurroundingNamespaces =
self.namespace->union(self.namespace.allSurroundingNamespaces)
```

Operáció (Operation)

Nincsenek kiegészítő szabályok.

Paraméter (Parameter)

[1] Egy ***Interfész*** nem használható úgy, mint ahogyan egy paraméter típusa.

```
not self.type.oclIsKindOf(Interface)
```

StruktúrálisTulajdonság (StructuralFeature)

[1] A kapcsolódó típusnak szerepelnie kell a kurrens ***Névterület***ben.

```
self.owner.namespace.allContents->includes(self.type)
```

.4 Szemantika

Ez a szakasz a Törzs csomagon belüli elemek dinamikus szemantikáját ismerteti. A szakasz felépítése a Törzs csomag fő konstrukcióján alapszik, mint például az interfész, az osztály és az asszociáció.

Öröklődés (Inheritance)

Az *öröklődés* értelmezéséhez először a *teljes deszkriptor* és a *szegmens deszkriptor* fogalmával kell tisztában lennünk. Egy *teljes deszkriptor* olyan teljes leírás, mely egy objektum, vagy más példány (ld. *Példányosítás*) leírásához szükséges. Tartalmazza az összes attributum, asszociáció és operáció leírását, melyek az objektumban megtalálhatók. Egy - az objektum-orientált "korszak" előtti - nyelvben az adatstruktúra teljes deszkriptora egészében közvetlenül volt deklarálva. Egy objektum-orientált nyelvben egy objektum leírása az inkrementális *szegmenseken* kívül helyezkedik el, mely szegmenseket az *öröklődés* felhasználásával kombinálták annak érdekében, hogy teljes leírást adjanak egy objektumhoz. A szegmensek azok a modellezési elemek, melyeket valójában a modellben deklaráltak; magukba foglalnak olyan elemeket, mint például az osztály, és más általánosítható elemek. Minden általánosítható elem tartalmaz egy tulajdonság-listát és más kapcsolatokat, melyeket azért fűzünk hozzájuk, hogy *örökíteni* lehessen őket az *őseikből*. Az *öröklődés* mechanizmusa definiálja, hogyan állnak elő a teljes deszkriptorok azon szegmensek halmazából, melyek az általánosítás révén kapcsolódnak. A teljes deszkriptorok implicitek, de definiálják az aktuális példányok struktúráját.

Az általánosítható elemek minden fajtájának van egy olyan halmaza, mely az *örökíthető tulajdonságokat* foglalja magába. Bármely modell elem számára ezek megszorításokat tartalmaznak. Az osztályozók számára ezek tulajdonságokat (attributumokat, operációkat, jelfogókat és metódusokat) és az asszociációs kapcsolatokban való részvételt tartalmaznak. Az általánosítható elem *ősei* - az összes ősével együtt (de a duplikátumok elhagyásával) - az ő szupertípusai.

Amennyiben egy általánosítható elemnek nincs szupertípusa, akkor az ő teljes deszkriptora azonos az ő szegmens deszkriptorával. Ha az általánosítható elemnek egy vagy több szupertípusa van, akkor az ő teljes deszkriptora tartalmazza azon tulajdonságok unióját, mely tulajdonságok az ő saját szegmens deszkriptorából és az ősök szegmens deszkriptoraiból származnak. Egy osztályozó számára ugyanolyan szignatúrával nem lehet deklarálni attributumot, operációt vagy jelet egynél több szegmensen belül (más szóval: nem újradefiniálhatók). Egy metódus több, mint egy szegmensben deklarálható; bármely szegmensben deklarált metódus felülbírálja, ill. felülírja ugyanazzal a szignatúrával, mely bármelyik ősben lett deklarálva. Ha kettő, vagy több metódus mégis megmarad, akkor ezek *konfliktusba (ütközésbe)* kerülnek és a modell kialakítása hibás lesz. A teljes deszkriptorok megszorításai magának a szegmensnek és az összes ősei megszorításainak uniója; amennyiben bármelyik közülük inkonzisztens, a modell kialakítása hibás lesz.

Bármely teljes deszkriptorban minden metódusnak kell lennie egy megfelelő operációjának az osztályozóhoz. Egy konkrét osztályozóban minden - teljes deszkriptorbei - operációnak kell lennie egy megfelelő metódusának a teljes deszkriptorban.

A teljes deszkriptor céljának ismertetése a *Példányosítás*nál megtalálható.

Példányosítás

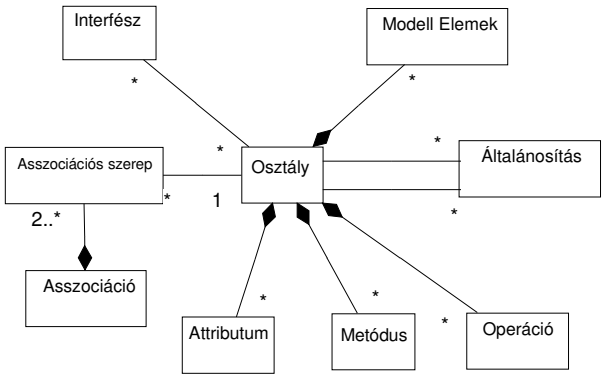
A modell célja, hogy leírja a rendszer és viselkedésének lehetséges állapotait. Egy rendszer állapota magába foglalja az objektumokat, az értékeket és a kapcsokat. Minden objektumot egy teljes osztály deszkriptor ír le; az osztály megfelel ennek a deszkriptornak, mely az objektum *közvetlen osztálya*. Hasonló módon minden egyes kapocs rendelkezik egy *közvetlen asszociációval* és minden értéknek van *közvetlen adattípusa*. Ezen példányok mindegyikére azt mondjuk, hogy az osztályozó közvetlen példánya, melytől az ő teljes deszkriptora származott. Egy példány *közvetett példánya* az osztályozónak vagy bármely ősének.

Egy objektum adattartalma magába foglal egy értéket (és semmi többet) minden attributumhoz az ő teljes osztály deszkriptorában; ennek az értéknek konzisztensnek kell lennie az attributum típusával. Egy kapocs adattartalma magába foglalja a példányok listáját, mely közvetett példánya minden résztvevő osztályozónak a teljes asszociáció deszkriptoron belül. A példányoknak és kapcsoknak engedelmeskedniük kell bármely megszorításnak a teljes deszkriptoron, melynek példányai (beleértve az explicit- és a beépített megszorításokat, mint amilyen egy multiplicitás).

Egy rendszer állapota egy *érvényes rendszer példány*, amennyiben minden példány közvetlen példánya valamely elemnek a rendszermodellben, és ha minden - a modell által kikényszerített - megszorítást kielégítenek a példányok.

Az UML viselkedési részei leírják az érvényes rendszerpéldányok érvényes sorrendjét, melyek mint a külső és belső viselkedési hatások eredményeiként fordulhatnak elő.

Osztály (Class)



Az *osztály* célja deklarálni metódusok, operációk és attributumok egy olyan csoportját, mely teljes mértékben leírja az objektumok felépítését és viselkedését. Az összes

objektum példányosítása egy osztályból történik, így lesz attributum értékük, mely illeszkedik az osztály deszkriptor attributumaival és támogatja azon operációkat, melyek az osztály teljes deszkriptorában találhatók. Néhány osztály közvetlenül nem példányosítható. Ezeket az osztályokat *absztrakt* osztályoknak mondjuk, és arra a célra használjuk őket, hogy más osztályok örökölhessék, ill. újra felhasználhassák az általuk deklarált tulajdonságokat.; nem lehet belőlük közvetlenül objektumot példányosítani, de egy objektum lehet közvetett példánya egy absztrakt osztálynak egy olyan alosztályon keresztül, amely nem absztrakt.

Amikor egy osztályt példányosítunk abból a célból, hogy új objektumot hozzunk létre, az előállt és inicializált új példány tartalmaz egy attributum értéket minden olyan attributumhoz, mely a teljes osztály deszkriptorban található. Az objektum inicializálása történhet a teljes osztály deszkriptor metóduslistájához való kapcsolódással is (Megjegyzendő: Egy tényleges implementáció úgy viselkedik, mintha teljes osztály deszkriptor lenne, de sok ügyes optimizáció lehetséges a gyakorlatban). Végül az új objektum azonosítója visszatér “alkotó” -jához. Egy jól kialakított rendszerben minden példány azonosítója egyedi és automatikus.

Egy osztálynak lehetnek *általánosításai* más osztályokra vonatkozólag. Ez annyit jelent, hogy egy osztály teljes osztály deszkriptorának származtatása a saját szegmens deklarációjából és azok őseiből történő örökítés révén történik. Az osztályok közötti általánosítás magába foglalja a helyettesíthetőséget, azaz egy osztály példánya mindig használható, amikor egy szuperosztály példánya szükséges. Ha egy osztályt gyökérként specifikáltak, akkor az nem lehet más osztályok alosztálya. Hasonlóképp, ha levélként specifikálták, akkor az adott osztály nem lehet más osztály szuperosztálya.

Minden *attributumnak*, melyet egy osztályban deklaráltak, láthatósága és típusa van. A láthatóság azt határozza meg, hogy az adott attributum publikusan rendelkezésre áll bármely osztály számára, vagy csak az adott osztályon és annak alosztályain belül használható (*védett*), ill. csak az adott osztályon használható (*privát*). Az attributum *célHatásköre* deklarálja, hogy értéke egy altípus, vagy saját típusának példánya legyen. Az attributum *tulajdonosiHatáskör*ére két alternatíva létezik: felállíthatja azon összes objektumot, melyek az adott osztályból (vagy alosztályból) lettek létrehozva és saját attributumértékkel rendelkeznek; vagy az értéket az osztály saját maga birtokolja. Egy attributum deklarálhatja még, hogy hány attributum értéknek kell kapcsolódnia a tulajdonosokhoz (*multiplicity*), kell -e lenniük kezdeti értékeknek, valamint ezek az attributum értékek megváltoztathatók -e. A következő esetek lehetségesek: nincsenek megszorítások (*none*), a meglévő érték nem módosítható, ill. nem vehetők fel új értékek, ha már inicializálták őket (*frozen*), vagy felvehetők új értékek, de nem távolíthatók el, és nem módosíthatók (*addOnly*).

Minden *operáció* számára az operáció név, a paraméterek típusai és a visszatérési típus(ok), valamint a láthatósága (ld. fentebb) meghatározott. Egy operáció tartalmazhatja még a meghívásával járó hatások specifikációját is. A specifikáció több különböző módon megadható: elő- és utófeltételekkel, pszeudokóddal, vagy szövegesen. Minden operáció deklarált, ha alkalmazható az osztály példányaihoz, vagy magához az osztályhoz (*tulajonosiHatáskör*). Továbbá az operáció megadja még, hogy az alkalmazás módosíthatja -e az objektum állapotát (*isQuery*). Az operáció azt is meghatározza, hogy az operáció realizálható -e egy alosztály különböző metódusai által (*isPolimorphic* [sokalakú]). Egy operációnak lehetnek *kiterjesztési pontjai* , melyek meghatározzák, hogy hol lehet további viselkedést hozzáilleszteni az operációhoz. A

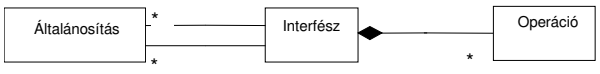
metódus egy operációt valósít meg, és ugyanolyan szignatúrája van, mint az operációnak; a törzsrésze implementálja az operáció specifikációját. Az utódokban a metódusok felülbírálják és lecserélik az ősöktől örökölt metódusokat (ld.: *Öröklődés*). Minden metódus egy - az osztályban deklarált, vagy egy őstöl örökölt - operációt implementál; ugyanazt az operációt csak egyszer lehet deklarálni egy teljes osztály deszkriptorban. A metódus specifikációjának illeszkednie kell a hozzá tartozó operáció specifikációjához, ahogyan azt az előbbiekben definiáltuk. Továbbá, ha egy operáció *isQuery* attributuma "Igaz [Igaz]", akkor bármely realizált metódusban is "Igaz [Igaz]" - nak kell lennie. Azonban ha ez az attributum hamis (*isQuery=false*) az operációban, és igaz a metódusban, nem szükséges, hogy az operáció módosítsa az állapotát. A láthatóság fogalma nem vonatkozik a metódusokra.

Az osztályoknak lehetnek *kapcsolataik* egymással. Ez magába foglalja azt, hogy a kapcsolódó osztályokból előállított objektumok szemantikusan kapcsolódnak, azaz az objektumok között kapcsok helyezkednek el az asszociációs kapcsolatok követelményeinek megfelelően. (Lásd: Asszociációs kapcsolat lentebb.) Az asszociációk az alosztályoktól öröklődnek.

Egy osztály interfészeket is megvalósíthat. Ez azt jelenti, hogy minden operáció, mely a teljes deszkriptorban található bármelyik realizált interfészhez, ugyanazzal a specifikációval kell jelen lennie a teljes osztály deszkriptorban. Az interfész és az osztály közötti kapcsolat nem szükségszerűen egy - egy kapcsolat; egy osztály felkínálhat több interfészt, valamint egy interfészt egy vagy több osztály kínálhat fel. Ugyanaz az operáció definiálható több interfészben, melyet egy osztály támogat; ha specifikációik azonosak, akkor nincs ütközés, egyébként a modell hibás lesz. Továbbá, egy osztály az interfészekben találhatók mellett tartalmazhat további operációkat is.

Egy osztály olyan szerepet játszik, mint a *névterület* az attributumokhoz, a kimenő szerepnevekhez az asszociációkon és az operációkhoz. Továbbá, egy osztály úgy viselkedik, mint egy a tartalmazott osztályok, interfészek és asszociációk (a hatáskörön belül definiált elemek; nem tartalmazzák az aggregációt) számára létrehozott névterület. A tartalmazott osztályozók felhasználhatók, mint egyszerű osztályozók a konténer osztályban. Azonban a tartalmakra nem hivatkozhat bárki a konténer osztályon kívülről. Ha egy osztály más osztály számára örökíti tartalmának láthatóságát, úgy, ahogyan azt a szuperosztályban definiálták, akkor ezek az elemek az alosztályban is láthatók lesznek. Ha egy elem láthatósága *public* vagy *protected*, akkor az alosztályban is látható lesz, de ha a láthatóság *private*, akkor az elem nem látható, ezért nem áll rendelkezésre az alosztály számára.

Interfész (Interface)



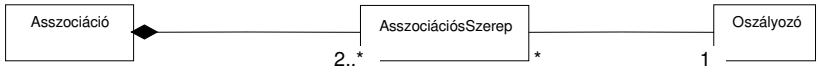
Egy interfész célja összegyűjteni azon operációkat, melyek az osztályozó által felkínált összefüggő szolgáltatást alkotják. Az interfészek eljárást biztosítanak arra, hogy az operációk csoportjait elkülönítsük és jellemezzük. Egy interfész csupán az operációk névvel ellátott csoportja, nem lehet közvetlenül példányosíttani. A példányosítható osztályozók, amilyenek az osztály, vagy a használati eset, interfészeket használhatnak a

példányaik által felkínált különböző szolgáltatások meghatározására. Több osztályozó is megvalósíthatja ugyanazt az interfészt; mindegyiküknek tartalmaznia kell legalább azokat az operációkat melyek az adott interfészben megtalálhatók. Egy *operáció* specifikációja tartalmazza az operáció szignatúráját, azaz a nevét, és a paramétereinek, valamint visszatérési értékének típusait. Egy interfész nem tartalmazza a megvalósító osztályozó belső struktúráját; pl. nem definiálja azt, hogy mely algoritmust kell használni egy operáció megvalósításához. Azonban egy operáció magába foglalhatja a meghívásával járó hatások specifikációját is. A specifikáció több különböző módon megadható: elő- és utófeltételekkel, pszeudokóddal, vagy szövegesen.

Minden operáció deklarálja, hogy alkalmazható -e annak az osztályozónak a példányaihoz, mely őt deklarálja, vagy magához az osztályozóhoz, azaz egy osztály konstruktora -e (*ownerScope*). Továbbá az operáció mindenkor megadja, hogy alkalmazása módosítja -e a példány állapotát (*isQuery*). Az operáció minden esetben azt is meghatározza, hogy minden osztályban ugyanúgy kell -e megvalósítani az adott operációt (*isPolymorphic*).

Egy interfész lehet más, az *általánosítások* által kifejezett más interfészek altípusa. Ez azt jelenti, hogy egy osztályozónak, melyet az interfész kínál fel, nem csak azokat az operációkat kell biztosítania, melyek az interfészben vannak deklarálva, hanem azon operációkat is, melyek az adott interfész őseiben voltak deklarálva. Ha az interfész gyökérként volt specifikálva, akkor nem lehet más interfészek altípusa. Hasonlóképp, ha levélként specifikálták, akkor nem lehet más interfész altípusa az adott interfésznek.xxx

Asszociáció (Association)



Egy *asszociáció* kapcsolatot (*kapcsot*) deklarál az összerendelt példányok között, azaz az osztályok között. Legalább két *asszociáció végpont*ot tartalmaz, mindegyik meghatároz egy kapcsolódó osztályozót és a tulajdonságok egy halmazát, melynek eleget kell tennie a kapcsolat érvényességi feltételeinek. Egy asszociáció végpont *multiplicitás* tulajdonsága meghatározza, hogy az adott végpontnál (a multiplicitás érték egy hordozójánál) osztályozó hány példánya kapcsolódhat az osztályozó egyetlen példányával más végpontnál. A *multiplicitás* a nemnegatív egészek egy tartománya. Az asszociáció végpont megadja még, hogy a kapcsolat kiterjedhet -e a példány felé, mely szerepet játszik a kapcsolatban (*isNavigable*), azaz a példány közvetlenül elérhető -e az asszociáción keresztül. Egy asszociáció végpont megadja még, hogy egy példány, mely szerepet játszik egy kapcsolatban felcserélhető -e más példányokkal. Megadhatja, hogy ne legyenek megszorítások (*none*), hogy a kapocs ne legyen módosítható, ha egyszer már inicializálták (*frozen*), vagy hogy lehessen új kapcsokat felvenni az asszociációhoz, de ne lehessen törölni, vagy módosítani őket (*addOnly*); e megszorítások nincsenek hatással azon objektumok módosíthatóságára, melyek a linkekhez kapcsolódnak. Ezenfelül a *célHatáskör* meghatározza, hogy az asszociáció végpontnak az osztályozó egy példányához vagy magához az osztályozóhoz kell kapcsolódnia. Az asszociáció végpont *isOrdered* attributuma megadja, hogy a példányok kapcsolatban vannak -e egy egyedülálló példánnyal más végpontnál, melynek fenntartott rendezettséggel kell rendelkeznie. Az új kapcsok beszúrásának sorrendjét azon operációkban kell

specifikálni, melyek felveszik vagy módosítják a kapcsokat. Megjegyzendő, hogy a rendezés egy végrehajtandó optimizálás és *nem* a logikailag rendezett asszociációkra vonatkozó példa, mert a rendezési információ egy rendezési műveletben nem nyújt semmilyen plusz információt.

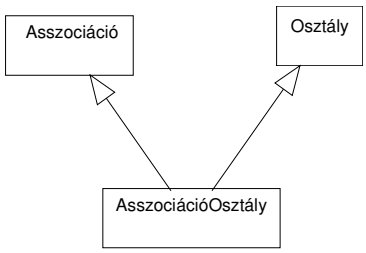
Egy asszociáció reprezentálhat egy *aggregációt*, azaz egy egész/rész kapcsolatot. Ebben az esetben az asszociáció végpont az egész elemhez kapcsolódik, mely kijelöli, és az asszociáció másik végpontja az aggregáció részét reprezentálja. Csak bináris asszociációk lehetnek aggregációk. A *kompozit* aggregáció az aggregáció egy erős formája, mely megköveteli, hogy egy kompozitban maximum egy rész példány legyen egyidejűleg, habár a tulajdonos ezt idővel megváltoztathatja. Továbbá egy kompozit magában foglal terjedési szemantikát, azaz néhány dinamikus szemantika képes tovább terjedni az egésztől a részeire. Például ha az egész másolódik, vagy törlődik, akkor a részek is hasonlóképp viselkednek. Egy *megosztott* aggregáció gyenge tulajdonosi viszonyt jelöl ki, azaz az egész szerepelhet több aggregátban, és tulajdonosa megváltoztathatja az idő során. Azonban a megosztott aggregáció szemantikája nem tartalmazza a részek törlését abban az esetben, amikor a konténerek egyike törlődik. Az aggregáció mindkét fajtája egy tranzitív, antiszimmetrikus kapcsolatot definiál, azaz a példányok egy irányított, körmentes gráfot alkotnak. A kompozíció példányok egy szigorú fát (vagy inkább erdőt) alkotnak.

Egy *minősítő* az összekapcsolt pédányok csoportjának egy partícióját deklarálja tekintettel a minősített végpontnál lévő példányra (a *minősített példány* annál a végpontnál van, melyhez a minősítő kapcsolódik). Egy minősítő példány minden minősítő attributumhoz tartalmaz egy értéket. Egy adott minősített objektum és egy minősítő példány, több objektum az asszociáció másik végpontjánál a deklarált multiplicitás által korlátozott. Általános esetben, melynél a multiplicitás 0..1, a minősítő érték egyedi a minősített objektumra való tekintettel, és éppenezért maximum egy kapcsolódó objektumot jelöl ki. A multiplicitás általános, 0..* esetében az összerendelt példányok halmaza olyan részhalmazokra bontható, melyeket egy adott minősítő példány szelektál. A multiplicitás 1 vagy 0..1 esetében a minősítőnek szemantikus és implementációs következménye egyaránt van; a multiplicitás 0..* esetében nincsenek valós szemantikus következmények, de javasolt, hogy egy implementáció, mely elősegíti az összerendelt példányok csoportjának elérését, egy adott minősítő érték által legyen kapcsolva.

Megjegyzendő, hogy egy minősítő multiplicitása adott azzal a feltételezéssel, hogy a minősítő érték rendelkezésre áll. A multiplicitás “nyílt”, ha nem feltételezzük azt, hogy a minősítő 0..*. Ez nem teljesen általános, de majdnem mindig megfelelő, mint olyan helyzetben, melyben a nyílt multiplicitás 1 lenne a legjobb a minősítő nélküli modellezésnél.

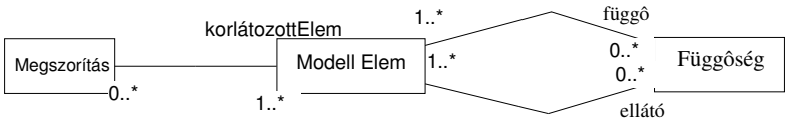
Megjegyzendő még, hogy egy minősített multiplicitás, melynek alsó határa, a nulla jelzi, hogy egy adott minősítő érték hiányozhat, amít az 1 -es alsó határ érték azt jelzi, hogy minden lehetséges minősítő értéknek jelen kell lennie. Az utóbbi, csupán olyan véges számú értékkel (amilyenek a felsorolt értékek, vagy az egész tartományok) rendelkező minősítőkhöz értelmezhető, melyek olyan teljes táblákat reprezentálnak, melyek indexelése néhány véges tartományú érték segítségével történik.

AsszociációOsztály [AssociationClass]



Egy asszociáció kifinomítható a saját tulajdonsághalmazához, azaz olyan tulajdonságokhoz, melyek nem tartoznak egyik kapcsolódó osztályozóhoz sem, hanem inkább magához az adott asszociációhoz. Az ilyen asszociációt *asszociáció osztály*nak nevezzük. Ez egy asszociáció a kapcsolódó osztályozókkal, és egyben osztály is, melynek tulajdonságai vannak, e tulajdonságokat más asszociációk is tartalmazzák. Az a fajta szemantika, mellyel egy asszociáció rendelkezik, nem más, mint egy közönséges asszociáció és egy osztály szemantikájának kombinációja.

Vegyes



A *megszorítás* egy logikai kifejezés egy vagy több elemre vonatkozólag, mely kifejezésnek mindig igaznak kell lennie. Egy megszorítást több különböző módon is megadhatunk, úgymint természetes nyelven, vagy a megszorítások nyelvezetén.

Egy *függőség* meghatározza, hogy a modell elemek egy halmaza megköveteli -e más modell elemek jelenlétét. Ez magában foglalja azt a tényt, hogy ha a forrás valamely körülmény miatt módosult, akkor a függőknek is valószínűleg módosulniuk kell. Ez azért van, mert a függőség több különböző módon specifikálható: természetes nyelv, vagy algoritmus használatával, de gyakran implicit módon is.

Az osztályozók speciális fajtája, mely hasonló az osztályhoz, az *adatTípus*, de az adattípus példányai primitív értékek, azaz nem objektumok. Például az egészek és a karakterláncok általában mint primitív értékek kezelődnek. Egy primitív értéknek nincs azonosítója, így ugyanannak az értéknek két előfordulása nem megkülönböztethető. Ezt általában egy attributum típusának specifikálására használjuk. Egy *felsorolt típus* egy felhasználó által definiált típus, mely az értékek véges számú halmazát tartalmazza.

.5 Szabványos elemek

A Törzs csomaghoz előre definiált sztereotípusok, megszorítások és csatolt értékek a következő táblázatban láthatók, definíciójuk a *Szabványos Elemek* függelékben olvasható.

Törzs - Szabványos Elemek

Modell Elem	Sztereotípus	Megszorítások	Csatolt Értékek
Asszociáció		implicit vagy (or)	
Attributum			folytonosság (persistence)
ViselkedésiTulajdonság	«create»(létrehoz) « destroy »(lerombol)		
Osztály (Class)	«implementationClass» « inherits »(örököl) « type »(típus)		
Osztályozó (Classifier)	« process »(folyamat) « stereotype » (sztereotípus) «utility»		elhelyezkedés (location) folytonosság (persistence) felelősség (responsibility) szemantika (semantics)
Megszorítások (Contraints)	« invariant » (állandó) « metaclass » (metaosztály) « postcondition (utófeltétel)» «powertype» « precondition » (előfeltétel)		
Elem (Element)			dokumentáció (documentation)
Általánosítás (Generalisaton)	« private » (privát) « subclass » (alosztály) « subtype » (altípus) « thread » (vezérszál) « uses » (használat)	teljes (complete) szétválasztott (disjoint) befejezetlen (incomplete) átfedő (overlapping)	
Operáció			szemantika (semantics)

.6 Megjegyzések

Az UML -ben az *Asszociáció*nak három különböző fajtája lehet: közönséges asszociáció, kompozit aggregát és megosztott aggregát. Mivel az aggregát szerkezetnek több különböző jelentése lehet az alkalmazási területtől függően, az UML egy pontosabb jelentést ad e két konstrukcióhoz, úgymint asszociáció és kompozit aggregát, és lazábban definiálva meghagyja a megosztott aggregátot.

Az *Operáció* egy fogalmi konstrukció, míg a *Metódus* implementációs konstrukció. A közös jellemzőik, mint a szignatúra birtoklása, a *ViselkedésiTulajdonság* metaosztályban van kifejtve, és az *Operáció*, valamint a *Metódus* konstrukciók meghatározott szemantikája a *ViselkedésiTulajdonság* megfelelő alosztályaiban vannak definiálva.

A *Használat* vagy *Kötés* függőség csak ugyanazon modellbeli két elem között jöhet létre, mivel a modell szemantikája nem függhet más modell szemantikájától. Ha különböző modellbeli elemek között jött létre kapcsolat, akkor a *Nyomkövetés*t, vagy a *Finomítás*t kell használni.

Az *AsszociációOsztály* konstrukció több különböző módon is kifejezhető a metamodellben, úgymint egy *Osztály* alosztálya, egy *Asszociáció* alosztálya, vagy mint egy *Osztályozó* alosztálya. Ugyanis egy *AsszociációOsztály* olyan konstrukció, mely egy asszociáció végpontokkal rendelkező asszociáció, és egyben egy osztály, mely tulajdonságokat deklarál. Kifejezésének legpontosabb módja egy *Asszociáció* és egy *Osztály* alosztályaként való kifejezés. E módon az *AsszociációOsztály* a másik két konstrukció összes tulajdonságát birtokolni fogja. Továbbá, ha a tulajdonságokat tartalmazó asszociációk új fajtái jelennek meg az UML -ben, ezek könnyen felvehetők lesznek mint az *Asszociáció* és a másik *Osztályozó* alosztályaiként.

Az *altípus* és *alosztály* kifejezések szinonímák, és azt jelentik, hogy az osztályozó egy példánya, mely más osztályozó altípusa, mindenhol használhatók, ahol az utóbbi osztályozó egy példánya kívánatos.

2 Kiegészítő elemek

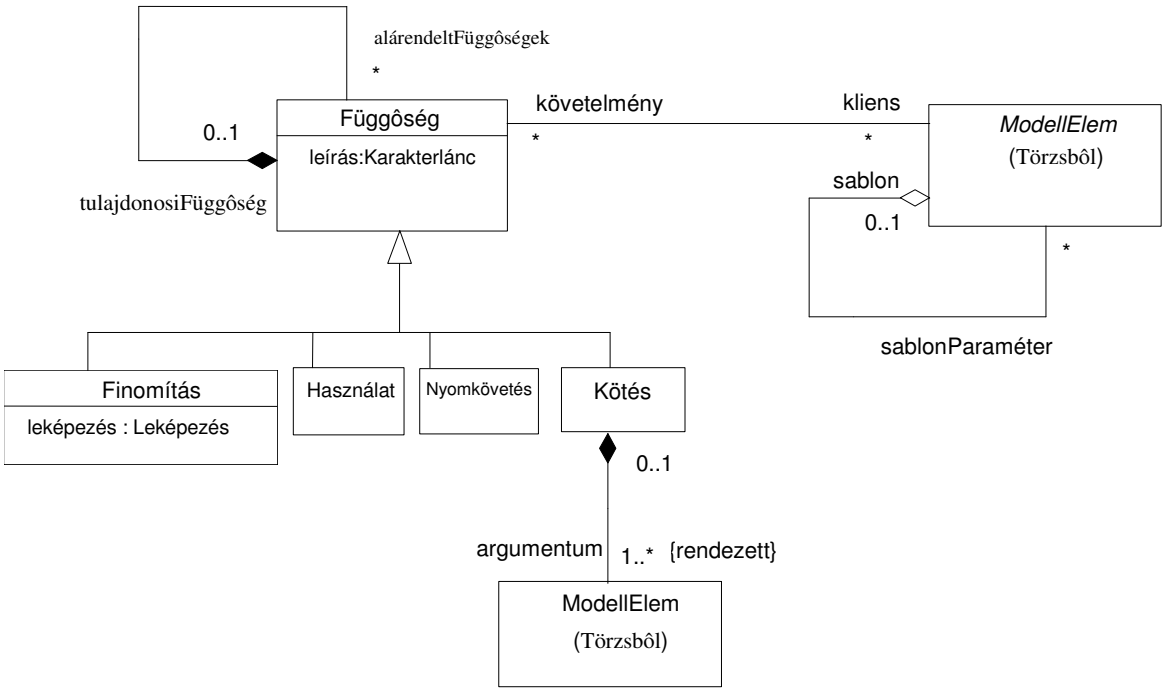
.1 Áttekintés

A kiegészítő elemek csomag az Alap csomag alcsomagja. További konstrukciókat definiál, melyek kiterjesztik a Törzs részt. A kiegészítő elemek infrastruktúrát biztosítanak a függőségekhez, sablonokhoz, fizikai struktúrákhoz és nézet elemekhez.

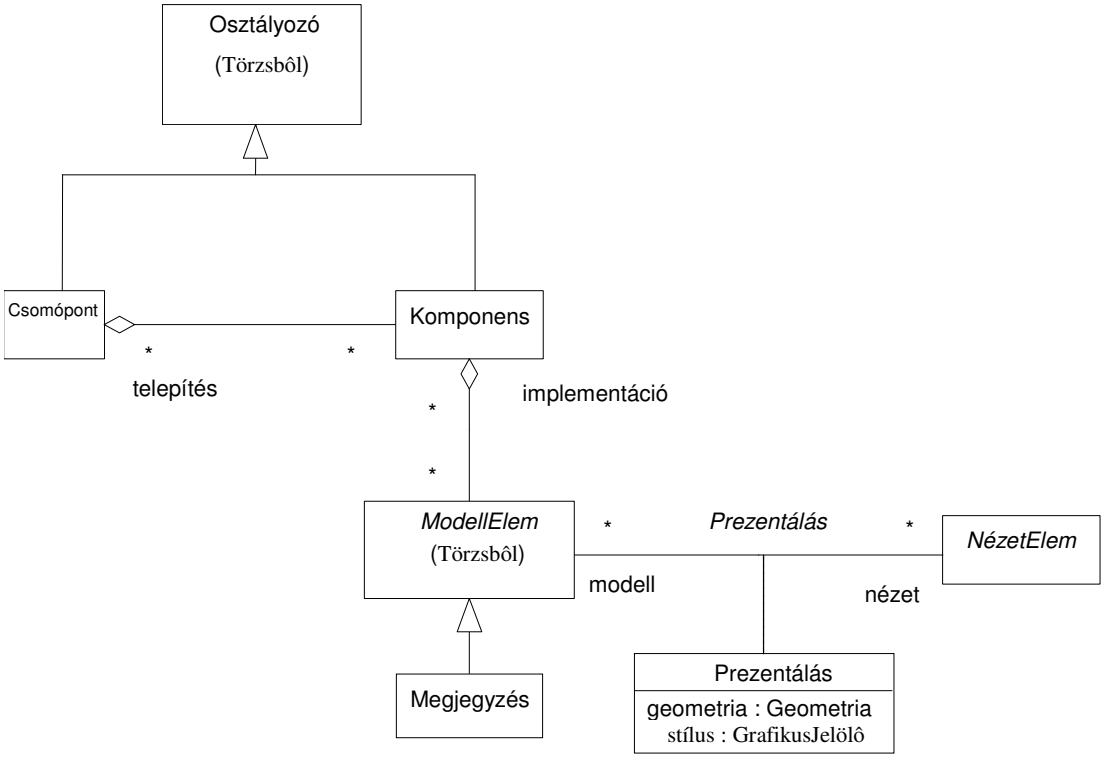
A következő szakaszok a kiegészítő elemek csomag absztrakt szintaxisát, jól képzett szabályait és szemantikáját írják le.

.2 *Absztrakt szintaxis*

A Kiegészítő Elemek absztrakt szintaxisa grafikusan látható a 7. és a 8. ábrán. A 7. ábrán azok a modell elemek láthatók, melyek a függőségeket és a sablonokat definiálják. A 8. ábra azokat a modell elemeket szemlélteti, melyek a fizikai struktúrát és a nézet elemeket definiálják.



7. ábra: Kiegészítő Elemek - Függőségek és Sablonok



8. ábra: Kiegészítő Elemek - Fizikai Struktúrák és Nézet Elemek

A Kiegészítő Elemek csomag a következő metaosztályokat tartalmazza:

Kötés (Binding)

A *kötés* kapcsolat egy sablon és a sablonból generált modell elem között. Magába foglalja azon argumentumok listáját, melyek illeszkednek a sablon paraméterekkel. A sablon egy olyan űrlap, melynek származtatása és módosítása helyettesítés által történik, mely egy implicit modellrészletet eredményez. Ez a modellrészlet úgy viselkedik, mintha a modell közvetlen része volna.

A metamodellben a *Kötés* egy *Függőség*, ahol az ellátó a sablon és a kliens a sablon példányosítása, mely végrehajtja egy sablon paramétereinek helyettesítését. A *Kötés*nek argumentumlistája van, mely a kliens kiszolgálása érdekében lecseréli az ellátó paramétereit. A kliens az ellátó paramétereinek kötése által teljes mértékben meghatározott és semmi plusz információt nem ad a sajátjaihoz.

Asszociációk (Associations)

argumentum (argument) Argumentumok rendezett listája. Mindent argumentum lecseréli a megfelelő ellátó paramétert az ellátó definícióban és az eredmény a kliens definícióját reprezentálja, mintha közvetlenül lett volna definiálva.

Megjegyzés (Comment)

A *Megjegyzés* egy a modell elemhez vagy modell elemek csoportjához kapcsolt magyarázó jegyzet.

A metamodellben egy *Megjegyzés* a *NézetElem* alosztálya. *ModellElemek* csoportjához kapcsolódik.

Komponens (Component)

A *komponens* egy újrafelhasználható rész, mely a modell elemek fizikai csomagokba való szervezését ("csomagolását") biztosítja.

A metamodellben a ***Komponens*** az ***Osztály*** alosztálya. A kapcsolódó specifikáció elemeinek fizikai csomagokba való szervezését ("csomagolását") biztosítja.

Asszociációk (Associations)

telepítés (deployment) ***Csomópontok*** halmaza, melyen a ***Komponens*** elhelyezkedik.

Függőség a Törzstől (Dependency from Core)

A *függőség* szemantikus kapcsolatot fejez ki maguk a modell elemek között (vagy inkább példányaik között), melyben egy elem megváltozása hatással lehet másik elem(ek) állapotára, illetve egy elem megváltozása megköveteli, maga után vonja más elem(ek) megváltozását.

A metamodellben a ***Függőség*** egy közvetlen kapcsolat a klienstől (vagy kliensektől) az ellágóig (vagy ellátókig), megadva, hogy a kliens függ az ellátótól, azaz az ellátó megváltozása hatással lehet a kliensre. A kapcsolat irányított, habár az irány figyelmen kívül hagyható a ***Függőség*** bizonyos altípusai esetében (amilyen a ***Nyomkövetés***).

Az összetartozó függőségek csoportosításához egy függőség funkcionálhat olyan konténerként, mely ***Függőség***ek csoportját foglalja magába. Ez hasznos, mivel gyakran léteznek függőségek elemek csoportjai között (mint amilyenek a ***Csomagok, Modellek, Osztályozók***, stb.). Például az egyik csomagnak a másikkal való függősége kiterjeszthető a két csomag elemeinek függőségeire.

Asszociációk (Associations)

kliens (client) Kliens. Az az elem, melyre az ellátó elem hatással van. Néhány esetben *(amilyen a **Nyomkövetés**)* az iránynak nincs jelentősége és csak a két elem megkülönböztetésére szolgál.

tulajdonosiFüggőség (owningDependency) Az alárendelt függőség (subDependency) inverze

alárendeltFüggőség (subDependency) Specifikusabb függőségek halmaza, mely kimunkál egy általánosabb függőséget.

ellátó (supplier) A **kliens** inverze. Kijelöli azt az elemet, melyre egy változtatás nincsen hatással. Egy kétutas kapcsolatban (amilyen néhány ***Finomítás***) ennek általánosabb elemnek kell lennie.

ModellElem (a Törzsből)

Egy *modell elem* olyan elem, mely a modellezett rendszerből képzett absztrakció. Ellentétben a *nézet elem*mel, melynek célja az emberi megértés elősegítése az információ szemléletes bemutatása által.

A metamodellben egy ***ModellElem*** egy modellbeli névvel ellátott egyed. Ez az alap az összes modellezési metaosztályhoz az UML -ben. Minden más modellező metaosztály a ***ModellElem***nek direkt vagy indirekt alosztálya.

Az összes ***ModellElem*** sablonnak tekinthető. Egy sablonnak *sablonParaméter*készlete van, mely megjelöli, hogy a ***ModellElem*** mely része képezi a sablon paramétereket. Egy ***ModellElem*** sablon, amikor legalább egy sablon paraméter létezik. Ha nem sablon, akkor, egy ***ModellElem***nek nem lehet sablon paramétere. Azonban az ilyen beágyazott paraméterek rendszerint nem teljesek, és nincsen szükség jól képzett szabályokra. Ezeket az argumentumokat alkalmazzuk, amikor a példányosított sablonnak jól kialakítottnak kell lennie. A részben példányosított sablonok engedélyezettek. Ez az az eset, amikor nem az összes, hanem csak néhány sablon paraméterhez biztosítunk argumentumot. Egy részben példányosított sablon még mindig sablon, mivel vannak paraméterei.

Asszociációk (Associations)

sablonParaméter (templateParameter) A sablon paraméterek egy rendezett listája. Minden paraméter egy ***ModellElem****et* jelöl ki a globális ***ModellElem*** hatókörén belül. A kijelölt ***ModellElem*** helyet biztosíthat egy valós ***ModellElem*** számára. Főként a sablon paraméter elemek nélkülözik a struktúrát. Például egy paraméter, mely egy ***Osztály***, nélkülözi a ***Tulajdonságokat***; melyek az aktuális argumentumban találhatók.

Csomópont (Node)

A *csomópont* egy futásidejű fizikai objektum, mely egy számítási erőforrást reprezentál, általában legalább memorizáló és gyakran feldolgozó képességgel is rendelkezik, azon komponensen, melyen alkalmazzák.

A metamodellben egy ***Csomópont*** az ***Osztály*** alosztálya. A ***Komponensek*** azon csoportjával áll kapcsolatban, melyek a ***Csomóponton*** tartózkodnak.

Asszociációk (Associations)

komponens (component) A ***Csomóponton*** tartózkodó ***Komponensek*** halmaza.

Prezentáció (Presentation)

A *prezentáció* a nézet elem és a modell elem (vagy valószínűleg mindegyikük halmaza) közötti kapcsolat. A részletek a grafikus szerkesztő eszköz implementációjától függenek.

A metamodellben a ***Prezentáció*** a ***ModellElem*** és a ***NézetElem*** közötti kapcsolatot testesíti meg, valamint biztosítja az elhelyezést és a prezentáció stílusát, melyet a ***ModellElem***ek prezentálásakor használunk.

Attributumok (Attributes)

geometria (geometry) A **NézetElem** kép geometriájának leírása.

stílus (style) A **NézetElem** képhez tartozó grafikus markerek leírása, mint például a szín, a textúra, font, vonalvastagság, árnyékolás, stb.

Finomítás (Refinement)

A *finomítás* modell elemek közötti kapcsolat különböző szemantikai szinteken, amilyenek az elemzés és tervezés.

A metamodellben a ***Finomítás*** egy ***Függőség***, ahol a kliensek az ellátótól származnak. Nem szükségszerű, hogy a származtatást algoritmus írja le; lehet, hogy emberi döntés szükséges a kliensek előállításához. A származtatás meghatározásának részletei túlmutatnak az UML hatáskörén, de megszorításokkal feltüntethető. A ***Finomítás*** felhasználható a modell lépésről-lépésre történő finomításához, optimizálásokhoz, transzformációkhoz, sablonokhoz, modell szintézishez, keretrendszer kompozícióhoz, stb.

Asszociációk (Associations)

leképezés (mapping) Két elem közötti leképezés leírása. A leképezés olyan kifejezés, melynek szintaxisa túlmutat az UML hatáskörén. Például a célja lehet egy karakterlánc reprezentálása.

Nyomkövetés (Trace)

A *nyomkövetés* egy fogalmi kapcsolat két elem, vagy elemek csoportjai között, mely egy fogalmat reprezentál különböző szemantikus szinteken vagy különböző nézőpontokból. Azonban nincs specifikus leképezés az elemek között. A konstrukció főként egy eszközként funkcionál a követelmények nyomon követésére. Hasznos ez a modellező számára is a különböző modellek változtatásainak nyomon követéséhez.

A metamodellben a ***Nyomkövetés*** különböző ***Modell***beli ***ModellElem***ek közötti ***Függőség*** ; a modellezett rendszer ua. részének elvonatkoztatása. A ***Nyomkövetések*** éppenezért inkább specifikációs szintbeli függőségeket jelölnek, mint futásidejű függőségeket. Következésképpen a ***Nyomkövetések*** ilyenformán nem magáról a rendszerről adnak információt, hanem inkább a rendszer ***Modell***jeiről. A függőség irányítottsága rendszerint figyelmen kívül hagyható.

Használat (Usage)

A *használat* olyan kapcsolat, melyben egy elem teljes implementációjához vagy operációjához más elemet (vagy elemek csoportját) igényli. Ez a kapcsolat nem pusztán történeti tény, hanem egy konkrét fennálló szükséglet. Éppenezért a *használattal* összerendelt két elemnek ugyanabban a modellben kell lennie.

A metamodellben a ***Használat*** egy ***Függőség***, melyben a kliens igényli az ellátó jelenlétét. Az, hogy a kliens hogyan haszálja az ellátót, ahogy egy osztály más osztály operációját hívja, egy metódusnak más osztálybeli argumentuma van, egy osztálybeli metódus, mely példányosítása egy másik osztályból történik, a ***Használat*** leírásában van definiálva.

NézetElem (ViewElement)

Egy *nézet elem* egy vagy több modell elem szöveges vagy grafikus megjelenítése.

A metamodellben egy ***NézetElem*** olyan ***Elem*** , mely a ***ModellElem***ek csoportját prezentálja az olvasó számára. Ez az alap minden UML -beli, prezentációhoz használt metaosztály számára. Minden más ilyen célú metaosztály a ***NézetElem*** nek direkt vagy indirekt alosztálya. A ***NézetElem*** egy absztrakt metaosztály. Ezen osztály alosztályai megfelelnek egy grafikus szerkesztő eszköznek és itt nincsenek specifikálva.

.3 Jól képzett szabályok

A következő jól képzett szabályokat alkalmazzuk a Kiegészítő Elemek csomaghoz.

Kötés (Binding)

[1] A ***ModellElem*** *argumentumnak* idomulnia kell a ***ModellElem*** *paraméterhez* egy Kötésen belül. Egy példányosításban ugyanolyan fajtájúnak kell lennie.

```
-- not described in OCL
```

Megjegyzés (Comment)

Nincsenek külön jól képzett szabályok.

Komponens (Component)

Nincsenek külön jól képzett szabályok.

Függőség (Dependency)

Nincsenek külön jól képzett szabályok.

További operációk

[1] Egy ***Függőség*** kompozit függőség, ha más függőségeket is tartalmaz.

```
isComposite : Boolean;
isComposite = (self.subDependency->size >= 1);
```

ModellElem (ModelElement)

Egy modell elem mindent *birtokol*, ami a kompozíciós kapcsolat által hozzá kapcsolódik.

A *sablon* egy modell elem, legalább egy sablon paraméterrel.

A modellnek azon része, melyet egy sablon birtokol, nem tartozik az összes jól képzett szabály hatásköre alá. Egy sablon nem használható közvetlenül egy jól kialakított modellben. Egy sablon kötésének eredményei a jól képzett szabályok hatásköre alá tartozik.

További operációk

[1] Egy ***ModellElem*** sablon, ha paraméterekkel rendelkezik.

```
isTemplate : Boolean;
isTemplate = (self.templateParameter->notEmpty)
```

[2] A ***ModellElem*** egy példányosított sablon, ha ***Kötési*** kapcsolat által áll összefüggésben egy sablonnal.

```
isInstantiated : Boolean;
isInstantiated = self.requirement->select(oclIsKindOf(Binding))->notEmpty
```

[3] A *sablonArgumentumok* egy példányosított sablon argumentumai, melyek helyettesítik a sablon paramétereket.

```
templateArguments : Set(ModelElement);
templateArguments = self.requirement->
         select(oclIsKindOf(Binding)).oclAsType(Binding).argument
```

Csomópont (Node)

Nincsenek külön jól képzett szabályok.

Prezentálás (Presentation)

Nincsenek külön jól képzett szabályok.

Finomítás (Refinement)

Nincsenek külön jól képzett szabályok.

Nyomkövetés (Trace)

[1] Egy ***Nyomkövetés*** ugyanazon ***Rendszer***beli két különböző ***Modell*** két ***ModellElem*** -csoportjához kapcsolódik.

```
self.client->forAll( e1, e2 | e1.model = e2.model ) and
self.supplier->forAll( e1, e2 | e1.model = e2.model ) and
self.client->asSequence->at (1).model <>
      self.supplier->asSequence->at (1).model and
self.client->asSequence->at (1).model.namespace =
      self.supplier->asSequence->at (1).model.namespace
```

Használat (Usage)

Nincsenek külön jól képzett szabályok.

NézetElem (ViewElement)

Nincsenek külön jól képzett szabályok.

.4 Szemantika

Valahányszor egy függőség ellátó eleme megváltozik, a kliens elem potenciálisan érvénytelenítve lesz. Ilyen érvénytelenítés után egy olyan ellenőrzést kell végrehajtani, melyet a származtatott kliens elemek lehetséges változásai követnek. Ezt az ellenőrzést olyan művelet után kell végrehajtani, mely képes megváltoztatni a származtatott elemet annak újbóli érvényesítése végett. Ennek az érvénytelenítésnek és változtatásnak a szemantikája túlmutat az UML hatáskörén.

Sablon (Template)

Egy fontos dinamikus következmény, hogy bármely modell elem, mely sablon, nem példányosítható. Csakis egy teljes mértékben példányosítható modell elemnek lehetnek példányai. Ez a szabály kifejezetten az osztályozó sablonokhoz alkalmazandó.

Az *űrlap* is sablon, de nem végső modell elem. Ez azért nem tartozik a normál jól képzett szabályok alá, mert szándékosan befejezetlen (kiegészítésre vár). Csak olyan sablon tartozik a jól képzett szabályok alá, mely argumentumok által kötött.

Egy további következmény, hogy egy sablonnak a modell olyan töredékét kell birtokolnia, mely nem része a végső effektív modellnek. Ha egy sablon kötött, a modell töredék, melyet birtokol implicit módon lemásolódik, a paramétereket lecserélik az argumentumok, és az eredmény implicit módon hozzáadódik az effektív modellhez, mintha az effektív modell modellezése közvetlenül történt volna.

NézetElem (ViewElement)

A *nézet elem* felelőssége biztosítani a modell elemek kollekciójának szöveges és grafikus vetületét. E kontextusban a vetület (projekció) azt jelenti, hogy a nézet elem egy ember által olvasható jelölést reprezentál a megfelelő modell elemekhez. E jelölés külön dokumentumban található.

A nézet elemeknek és a modell elemeknek egyezniük kell, de az ehhez szolgáló mechanizmusok tervezésének problematikája a modell szerkesztő eszközök kompetenciájába tartozik.

.5 *Szabványos elemek*

A Kiegészítő Elemek csomaghoz előre definiált sztereotípusok, megszorítások és csatolt értékek a következő táblázatban láthatók, és definíciójuk a *Szabványos Elemek* függelékben található.

Modell Elem	**Stereotípus**	**Megszorítás**	**Csatolt Érték**
Megjegyzés [Comment]	«követelmény (requirement)»		elhelyezkedés (location)
Komponens [Component]	«dokumentum (document)» «végrehajtható (executable)» «fájl (file)» «barát (friend)» «könyvtár (library)» «táblázat (table)»		
Függőség [Dependency]	«megfelelés (becomes)» «hívás (call)» «másolás (copy)» «törlés (deletion)» «származtatott (derived)» «import» «példány (instance)» «metaosztály (metaclass)» «powertype» «küldés (send)»		

Kiegészítő Elemek - Szabványos Elemek

3 Kiterjesztési mechanizmusok

.1 Áttekintés

A Kiterjesztési Mechanizmusok csomag az Alap csomag alcsomagja, mely specifikálja a modell elemek testreszabásának és új szemantikával való kiterjesztésének módját. Szemantikát definiál a sztereotípusokhoz, megszorításokhoz és a csatolt értékekhez.

Az UML számos modellezési fogalmat és jelölést biztosít, melyek tervezésekor gondoskodtak arról, hogy azok találkozzanak a tipikus szoftver modellezési projektek kívánalmaival. Azonban a felhasználók néha további tulajdonságokat és/vagy jelöléseket igényelhetnek, melyek túlmutatnak az UML szabványon. Továbbá a felhasználók gyakran kívánnak kiegészítő, nem szemantikus infromációt fűzni a modellekhez. Ezek az igények a három beépített kiterjesztési mechanizmus által találkoznak az UML -ben, melyek lehetővé teszik újfajta modellezési elemek hozzáadását a modellezők repertoárjához, csakúgy, mint a kötetlen formájú információk hozzáfűzését a modellezési elemekhez. E három kiterjesztési mechanizmus használható külön-külön, és együtt is az új modellezési elemek definiálásához, melyek különböző szemantikájuk, karakterisztikájuk és jelölésük lehet, összefüggésben a beépített UML modellezési elemekkel, melyeket az UML metamodell specifikál. A konkrét szerkezetek a Kiterjesztési Mechanizmusokban vannak definiálva. Ide tartoznak a ***Megszorítás, Sztereotípus*** és a ***CsatoltÉrték*** .

Az UML kiterjesztési mechanizmusai az alábbi célokat szolgálják:

- Felhasználhatók új modellezési elemek felvételéhez az UML modellek létrehozásakor.

- Az UML specifikációban is használatosak olyan szabványos tételek definiálásához, melyek nem elég számottevőek, vagy összetettek ahhoz, hogy közvetlenül definiáljuk őket mint UML metamodell elemeket.

- Felhasználhatók a feldolgozás specifikus vagy implementációs nyelv specifikus kiterjesztések definiálásához.

- Felhasználhatók önkényes szemantikus és nem szemantikus információk modell elemekhez való hozzáfűzéséhez.

A legfontosabb, hogy a beépített kiterjeszséti mechanizmusok a *Sztereotípus* fogalmon alapulnak. A sztereotípusok a modell elemek objektum modell szinten való osztályozásának módját biztosítják és “virtuális” UML metaosztályok felvételét segítik elő új metaattributumokkal és szemantikával. Más beépített kiterjesztési mechanizmusok a tulajdonság lista elképzelésen alapulnak, mely lista *címkékből, értékekből* és *megszorítások*ból áll. Mindezek lehetővé teszik a felhasználók számára

további tulajdonságok és szemantika direkt módon történő hozzáadását az egyes modell elemekhez, csakúgy, mint a Sztereotípus által osztályozott elemekhez.

A sztereotípus egy UML modell elem, mely más UML elemek osztályozására (vagy megjelölésére) használunk, ezért néhány esetben úgy viselkednek, mintha új "virtuális" vagy "pszeudo" metamodell osztályok példányai lennének, melyek formája a már létező "alap" osztályén alapul. A sztereotípus így tehát megnöveli az UML beépített metamodell osztályhierarchiáján alapuló osztályozási mechanizmust, és éppenezért az új sztereotípusok nevei nem ütköznek az előre definiált metamodell elemek vagy egyéb sztereotípusok neveivel. Bármely modell elemet megjelölhet legfeljebb egy sztereotípus, de bármely sztereotípus összeállítható több más sztereotípus specializációjaként.

Egy sztereotípus bevezethet újabb értékeket, további megszorításokat és új grafikus megjelenítést. Minden modell elem, melyet egy bizonyos sztereotípus osztályoz ("sztereotípizál") megkapja ezeket az értékeket, megszorításokat és megjelenítést. A sztereotípusokhoz kapcsolt grafikus megjelenítés lehetővé teszi a felhasználók számára, hogy új módszereket vezessenek be a sztereotípusok által osztályozott modell elemek megkülönböztetésére.

A sztereotípus megosztja alap osztályának attributumait, asszociációit és operációit, de rendelkezhet további jól képzett megszorításokkal, csakúgy mint különböző jelentésű értékekkel. A szándék az, hogy egy eszköz vagy tároló képes legyen befolyásolni egy sztereotípizált elemet ugyanúgy, mint egy közönséges elemet a legtöbb szerkesztési és tárolási célhoz, amíg megkülönbözteti azt bizonyos szemantikus operációkhoz, mint amilyen a jól kialakított ellenőrzés, kódgenerálás vagy riport írás.

Bármely modellezési elem rendelkezhet önkényesen hozzákapcsolt információval a tulajdonság lista űrlapjában, mely címke-értékpárokból áll. Egy ilyen címke egy név karakterlánc, mely egyedi egy megadott elemre nézve. Ez a megadott elem egy kapcsolódó tetszőleges értéket jelöl ki. Az értékek tetszőlegesek lehetnek, de az egységes információcsere érdekében karakterláncként kell őket megadni. A címke egy tetszőleges tulajdonság nevét reprezentálja a megadott értékkel. A címkék felhasználhatók menedzsment információk megjelenítéséhez (*szerző, esedékesség dátuma, státusz*), kódgenerálási információhoz (*optimizálásiSzint, konténerOsztály*), vagy további szemantikus infromációhoz, melyet egy adott sztereotípus megkíván.

Lehetőség van címkék listájának specifikálására (alapértelmezett értékekkel, amennyiben ez szükséges), mely címkék egy bizonyos sztereotípushoz szükségesek. Az ilyen megkövetelt címkék a sztereotípus "pszeudoattributumaiként" funkcionálnak a valós attributumok kiegészítése céljából. A valós attributumokat az alap elem osztály szolgáltatja. Az ilyen címkékhez engedélyezett értékek halmaza korlátozható.

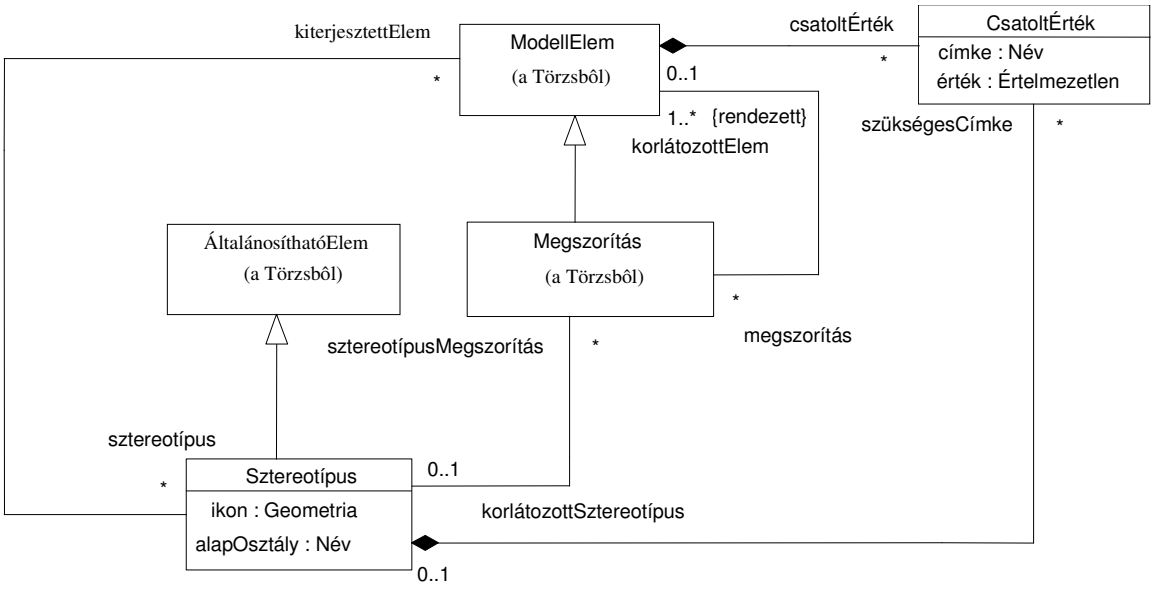
Egy modell elem egyedi megadásához nem szükséges azt rendezetten sztereotípizálni. Egy modell elemhez (akár sztereotípizált, akár nem) - szemantikájának megváltoztatása érdekében - közvetlenül is hozzáfűzhetők megszorítások. Hasonlóan, egy tulajdonság lista, mely címke érték-párokból áll, közvetlenül hozzáilleszthető bármely modellelemhez. A tulajdonság lista csatolt értékei lehetővé teszik, hogy karakterisztikát rendeljünk a modell elemekhez egy rugalmas, egyedi alapon. A címkék a felhasználó által definiálhatók; de némelyikük előre definiált, és ezek a *Szabványos Elemek* c. függelékben megtalálhatók.

A megszorítások vagy a csatolt értékek bizonyos szeteorípusokkal állnak kapcsolatban, mely sztereotípusokat a modell elemek szemantikájának kiterjesztéséhez használjuk. A szóbanforgó modell elemeket az adott sztereotípus osztályozza. A megszorításokat minden - az adott sztereotípus által megjelölt - modell elemnek vizsgálnia kell.

A következő szakaszok a Kiterjesztési Mechanizmusok csomag absztrakt szintaxisát, jól képzett szabályait és szamantikáját írják le.

.2 *Absztrakt szintaxis*

A Kiterjesztési Mechanizmusok csomag absztrakt szintaxisa grafikus formában a 9. ábrán látható.



9. ábra: Kiterjesztési Mechanizmusok

A Kiterjesztési Mechanizmusok csomag a következő metaosztályokat tartalmazza:

Megszorítás (Constraint)

A *megszorítás* fogalom lehetővé teszi, hogy nyelvi szempontból új szemantikát specifikáljunk egy modell elemhez. A specifikációt mint kifejezést írjuk le egy erre a célra szolgáló megszorítás nyelven. A nyelv különösen megszorítások írásához (mint amilyen az OCL), programozási nyelvhez, matematikai jelölésekhez alkalmas. Ha a megszorításokat egy modell szerkesztő eszköz érvényesíti, akkor az eszköznek értelmeznie kell a megszorítás nyelv szintaxiság és szemantikáját. Mivel a nyelv kiválasztása önkényes, a megszorítások kiterjesztési mechanizmusnak tekintendők.

A metamodellben egy ***Megszorítás*** közvetlenül hozzákapcsolható egy ***ModellElem***hez, hogy leírja azon szemantikus korlátozásokat, melyeknek engedelmeskednie kell e ***ModellElem***nek. Bármely ***Megszorítás*** hozzákapcsolható egy ***Sztereotípushoz*** , mely minden olyan ***ModellElem***hez alkalmazunk, mely hordozza a megadott ***Sztereotípust.***

Attributumok (Attributes)

törzs (body) Egy logikai kifejezés, mely a megszorítást definiálja. A kifejezést karakterlánc formátumban írjuk egy kijelölt nyelven. Ahhoz, hogy a modell kialakítása megfelelő legyen, a kifejezésnek mindig igaz értéket kell szolgáltatnia amikor a megszorítás alá eső elemek példányai számára kiértékelődnek bármely olyan időpillanatban, amikor a rendszer stabil (azaz nem egy atomi operáció végrehajtása közben).

Asszociációk (Associations)

korlátozottElem (constrainedElement) A megszorítás alá tartozó elemek rendezett listája. A megszorítás az ő példányaihoz alkalmazott.

korlátozottSztereotípus (constrainedStereotype) A megszorítás alá tartozó sztereotípusok rendezett listája. A megszorítás a sztereotípus által osztályozott elemek példányaihoz alkalmazott.

Bármely egyedi megszorításnak vagy egy ***korlátozottElem*** kapcsa (linkje) vagy egy ***korlátozottSztereotípus*** kapcsa van, de mindkettőt nem birtokolhatja.

ModellElem (mint kiterjesztett)

Bármely modell elem rendelkezhet tetszőleges csatolt értékekkel és megszorításokkal. Egy modell elemnek maximum egy sztereotípusa lehet, mely alaposztályának illeszkednie kell a modellezési elem (amilyen az Osztály, Asszociáció, Függőség, stb.) UML osztályához. Egy sztereotípus jelenléte implicit megszorításokat kényszeríthet ki a modellezési elemen és megkövetelheti meghatározott csatolt értékek jelenlétét.

Asszociációk (Associations)

megszorítás (constraint) Egy megszorítást, mint feltételt a modell elem példányai számára kell kielégíteni. Egy modell elemnek lehet megszorítás csoportja. Egy megszorítás csak a rendszer stabil állapotában értékelődhet ki, azaz nem egy elemi operáció közepén.

sztereotípus (stereotype) Maximum egy sztereotípus jelölhető ki, mely tovább minősíti az adott modell elem UML -beli osztályát (az *alap osztályt*). A sztereotípus nem változtatja meg az alaposztály struktúráját, de specializálhatja azt további megszorításokkal és csatolt értékekkel. Minden - a sztereotípuson levő - megszorítás és csatolt érték alkalmazható azon modell elemekhez, melyeket az adott sztereotípus osztályoz. A sztereotípus úgy viselkedik, mint egy “pszeudo metaosztály”, mely a modell elemet írja le.

csatoltÉrték (taggedValue) Egy - a modell elemhez kapcsolt - tetszőleges tulajdonság. A csatolt érték értelmezése túlmutat az UML metamodell kiterjedési körén. Egy modell elemnek lehet csatolt érték-halmaza, de egy egyedülálló modell elemnek legfeljebb egy csatolt értéke lehet egy megadott címke névvel. Ha a modell elemnek van sztereotípusa,

akkor ez a sztereotípus előírhatja bizonyos címkék meglétét, melyek az alapértelmezett értékeket biztosítják.

Sztereotípus (Stereotype)

A *sztereotípus* fogalom eljárást biztosít az elemek osztályozásához (megjelöléséhez), ezáltal ezen elemek bizonyos tekintetben úgy viselkednek, mintha új "virtuális" metamodell konstrukciók pédányai lennének. A példányoknak ugyanolyan struktúrájuk (attributumok, asszociációk, operációk) van, mint ugyanazon fajta nem sztereotípizált példányainak; a sztereotípus meghatározhat további megszorításokat és szükséges csatolt értékeket, melyek a példányokhoz alkalmazandók. Mindemellett egy sztereotípus arra is felhasználható, hogy két azonos struktúrájú elem jelentése vagy használata közötti különbséget jelezzük.

A metamodellben a ***Sztereotípus*** metaosztály az ***ÁltalánosíthatóElem*** egyik altípusa. A ***Sztereotípushoz*** illesztett ***CsatoltÉrtékeket*** és a ***Megszorítás***okat a ***Sztereotípus*** által osztályozott ***ModellElem***ekhez alkalmazzuk. Egy sztereotípus egy geometriai ikont is specifikálhat az elemek sztereotípussal együtt történő prezentálásához.

A ***Sztereotípusok ÁltalánosíthatóElem***ek. Ha egy sztereotípus egy másik sztereotípus altípusa, akkor minden megszorítást és csatolt értéket örököl a sztereotípusának szupertípusától és az alaposztály ugyanazon fajtájához kell alkalmazni. Egy sztereotípus nyomon követi azt az alaposztályt, melyhez alkalmazható.

Attributumok (Attributes)

alapOsztály (baseClass) Specifikálja egy UML modellezési elem nevét, melyhez a sztereotípust alkalmazzuk. Ilyenek az ***Osztály, Asszociáció, Finomítás, Megszorítás,*** stb. Ez a metaosztály neve, azaz inkább egy UML metamodellbeli osztály maga, mint egy felhasználói modell osztály.

ikon (icon) Geometriai leírás egy ikonhoz, melyet a sztereotípus által osztályozott modell elem képének prezentálásához használunk.

Asszociációk (Associations)

kiterjesztettElem (extendedElement) Kijelöli azt a modell elemet, melyre a sztereotípus hatással van. Mindegyiknek az *alapOsztály* attributuma által meghatározott fajtájúnak kell lennie.

sztereotípusMegszorítás (stereotypeConstraint) Azon megszorításokat jelöli ki, melyeket az adott sztereotípust hordozó elemekhez alkalmazunk.

szükségesCímke (requiredTag) Csatolt értékek halmazát specifikálja, mely mindegyike meghatároz egy címkét. Ez a címke olyan elem, melyet a megkívánt sztereotípus osztályoz.

CsatoltÉrték (TaggedValue)

A csatolt érték egy (Címke, Érték) pár tetszőleges információ hozzáfűzését engedélyezi bármely modell elemhez. Egy címke egy tetszőleges név; néhány címke név előre

definiált a Szabványos Elemekben (ld. ott). Legfeljebb egy névvel ellátott csatolt értékpár kapcsolható egy adott modell elemhez. Más szavakkal, létezik a címke karakterláncok által kiválasztott értékeknek egy táblázata, mely hozzákapcsolható bármely modell elemhez.

Egy címke értelmezése (szándékosan) túlmutat az UML hatáskörén; a felhasználónak, vagy az eszköz konvenciónak kell meghatározni azt. Várható, hogy a különböző modell elemző eszközök címkéket fognak definiálni olyan információk megadásához, melyek az UML alap szemantikáján túlmutató operációikhoz szükségesek. Ilyen információhoz tartozik a kódgenerálási opció, modell menedzsment információ, vagy felhasználó specifikus kiegészítő szemantika.

Attributumok (Attributes)

címke (tag) Olyan név, mely a **ModellElem**ekhez hozzáfűzendő kiterjeszthető tulajdonságot jelöl. A címke nevek számára fenntartott hely van. Az UML nem definiál mechanizmust a név nyilvántartáshoz, de a modell szerkesztő eszközöktől elvárható, hogy biztosítsanak ilyen szolgáltatást. Egy modell elemnek legfeljebb egy csatolt értéke lehet egy megadott névve. Valójában egy címke egy pszeudoattributum, mely hozzákapcsolható a modell elemekhez.

érték (value) Egy tetszőleges érték. Az értéknek az egységes kezelhetőség érdekében karakterláncként kifejezhetőnek kell lennie. A megengedhető értékek tartománya a felhasználó vagy az eszköz által a címkéhez alkalmazott interpretációtól függ; specifikációja túlmutat az UML hatáskörén.

Asszociációk (Associations)

csatoltÉrték (taggedValue) A ***CsatoltÉrték*** az, amit egy ***ModellElem****hez* hozzáfűzünk.

szükségesCímke A ***CsatoltÉrték*** az, amit egy ***Sztereotípushoz*** hozzáfűzünk. Egy adott ***CsatoltÉrték*** vagy egy ***ModellElem****hez*, vagy egy ***Sztereotípushoz*** kapcsolható, de mindkettőhöz nem.

.3 Jól képzett szabályok

A következő jól képzett szabályokat alkalmazzuk a Kiterjesztési Mechanizmusok csomaghoz:

Megszorítás (Constraing)

[1] Egy *Sztereotípushoz* kapcsolt *Megszorítás* nem ütközhet azon megszorításokkal, melyek bármely örökölt sztereotípuson érvényesek, vagy az *alapOsztállyal* kapcsolódnak.

```
-- cannot be specified with OCL
```

[2] Egy sztereotípizált ModellElemhez kapcsolt megszorítás nem ütközhet egyik kapcsolt osztályozó sztereotípus megszorításával sem, és szintén nem ütközhet a Modell Elem Osztályával (*alapOsztály*).

```
-- cannot be specified with OCL
```

[3] Egy sztereotípushoz kapcsolt megszorítást a későbbiekben minden ModellElemhez alkalmazni fogunk, melyet az adott sztereotípus osztályoz, és nem ütközhet a kapcsolt osztályozó sztereotípusok megszorításaival, sem a Modell Elem Osztályával (*alapOsztály*).

```
-- cannot be specified with OCL
```

Sztereotípus (Stereotype)

[1] A *Sztereotípus* nevek nem ütközhetnek egyik *alapOsztály* névvel sem.

```
Stereotype.oclAllInstances->forAll(st | st.baseClass <> self.name)
```

[2] A *Sztereotípus* nevek nem ütközhetnek egyik örökölt Sztereotípus nevével sem.

```
self.allSupertypes->forAll(st : Stereotype | st.name <> self.name)
```

[3] A *Sztereotípus* nevek nem ütközhetnek az (M2) metaosztály névterületén belül, sem bármely örökölt sztereotípus nevével, sem pedig bármely *alapOsztály* névvel.

```
-- M2 level not accessible
```

[4] Az *alapOsztály* névnek biztosítottnak kell lennie; az *ikon* opcionális egy speciális implementáción belül.

```
self.baseClass <> ''
```

[5] Egy sztereotípushoz kapcsolt címke nevek nem ütközhetnek a megfelelő *alapOsztály* elem M2 metaattributum névterületével, sem bármely örökölt sztereotípus címke nevével.

```
-- M2 level not accessible
```

ModellElem (ModelElement)

[1] Egy ModellElemmel (közvetlenül egy tulajdonságlistán keresztül vagy indirekt módon egy sztereotípuson keresztül) összekapcsolt címkék nem ütközhetnek egyik - ModellElemmel összekapcsolt - meta-attributummal sem.

```
-- not specified in OCL
```

[2] Egy modell elemnek legfeljebb egy csatolt értéke lehet egy adott címkenévvel.

```
self.taggedValue->forAll(t1, t2 : TaggedValue |
  t1.tag = t2.tag implies t1 = t2)
```

[3] Ha T modellElem.sztereotípus.szükségesCímke olyan, hogy T.érték = **határozatlan** akkor a modellElemnek rendelkeznie kell egy olyan csatolt értékkel, ahol név = T.név .

```
self.stereotype.requiredTag->forAll(tag |
      tag.value = Undefined implies self.taggedValue->exists(t |
```

```
t.tag = tag.tag))
```

CsatoltÉrték (TaggedValue)

Nincsenek külön jól képzett szabályok.

.4 Szemantika

A megszorításokat, sztereotípusokat és a csatolt értékeket a modell elemekhez és nem a példányokhoz alkalmazzuk. Magához a modellezési nyelvhez és nem a futásidejű környezethez nyújtanak kiterjesztéseket. Kihatással vannak a modell struktúrájára és szemantikájára. E fogalmak "metaszintű" kiterjesztéseket reprezentálnak az UML-hez; azonban nem tartalmazzák egy "nehézsúlyú" metamodell kiterjesztési nyelvezet teljes képességét, és úgy tervezték őket, hogy az eszközök nem igénylik a metaszintű szemantika implementációját.

Egy modellen belül bármely felhasználószintű modell elemnek lehet megszorítás, ill. csatolt érték halmaza. A megszorítások a modell példányosításán érvényesítendő korlátozásokat specifikálják. Egy felhasználó szintű modell elem példányának ki kell elégítenie minden - a modell elemen érvényesítendő - megszorítást ahhoz, hogya modell kialakítása megfelelő legyen. A megszorítások kiértékelésének akkor kell megtörténnie, amikor a rendszer stabil állapotban van, azaz miután bármilyen belső operáció befejeződött és a rendszer külső eseményekre vár. A megszorítások külön e célra kifejlesztett megszorítás nyelven íródnak, mint amilyen az OCL, C++, vagy akár a természetes nyelv is. A megszorítások interpretációját a megszorítás nyelvvel kell specifikálni.

Egy felhasználó szintű modell elemnek legfeljebb egy csatolt értéke lehet egy adott címke névvel. Minden címkenév egy felhasználó által definiált tulajdonságot reprezentál. E tulajdonság modell elemekhez alkalmazható egy egyedi értékkel. A címke jelentése túlmutat az UML kiterjedési körén, a felhasználók és modell elemző eszközök közötti konvencióknak kell meghatározniuk.

Az a szándék, hogy a megszorítások és a csatolt értékek egyaránt karakterláncként legyenek megadva, így azok könnyen szerkeszthetők, tárolhatók, ill. áthelyezhetők azon eszközök által, melyek nem képesek értelmezni szemantikájukat. Az elképzelés az, hogy a szemantika értelmezése lokalizálható legyen néhány modulra, melyek használják az adott értékeket. Például egy kódgenerátor használhatná a csatolt értékeket a kódgenerálási folyamat kialakítására; egy folyamat tervező eszköz felhasználhatná a csatolt értékeket a modell elemek tulajdonosi viszonyainak és állapotának megjelenítéséhez. Más modulok egyszerűen és változtatás nélkül megőrizhetnék az értelmezetlen értékeket (karakterlánc formátumban).

Egy sztereotípus egy *alapOsztály*ra utal, mely egy osztály az UML metamodellben (nem felhasználó szintű modellezési elelm). Ilyenek az ***Osztály***, ***Asszociáció***, ***Finomítás*** , stb. Egy sztereotípus egy vagy több létező sztereotípusnak (melyeknek mind ugyanarra az alaposztályra, vagy alapOsztályokra kell utalniuk, mely ua. alapOsztálytól származik) lehet altípusa, mely esetben örökli a megszorításaikat és szükséges

címkéiket, valamint hozzátehet továbbiakat a sajátjaihoz. Egy sztereotípus hozzátehet új megszorításokat, új ikont a vizuális megjelenítéshez, valamint az alapértelmezett értékek listáját is felveheti.

Amennyiben egy felhasználó szintű modell elemet egy kapcsolt sztereotípus osztályoz, akkor a modell elem UML alaposztályának illeszkednie kell a sztereotípus által specifikált alaposztállyal. Bármely - a sztereotípuson érvényesítendő - megszorítás implicit módon kapcsolódik a modell elemhez. Bármely csatolt érték a sztereotípuson implicit módon kapcsolódik a modell elemhez; ha az értékek bármelyike ***határozatlan***, akkor a modell elemnek egyértelműen definiálnia kell csatolt értékeket ua. címke névvel. Ellenkező esetben a modell kialakítása nem lesz megfelelő. Ha a sztereotípus egy vagy több más sztereotípus altípusa, akkor bármely megszorítás vagy csatolt érték is alkalmazható azon sztereotípusokból a modell elemhez (mivel ezen sztereotípus által öröklődtek). Ha bármilyen ütközés fordul elő a többszörös megszorítások vagy csatolt értékek (örökölt vagy közvetlenül specifikált) között, akkor a modell kialakítása nem lesz megfelelő.

.5 *Szabványos elemek*

Jegyzet.

.6 *Megjegyzések*

A nézet egy implementációs pontjából a sztereotípizált osztály példányai úgy tárolódnak, mint az alaposztály példányai a sztereotípus névvel mint tulajdonsággal. A csatolt értékeket (karakterlánccal kifejezett) értékek táblázataként (minősített asszociáció) lehet és kell implementálni. Ezen értékeket a (karakterláncként reprezentált) címke nevek szelektálják. Az UML metamodell osztályok és címke nevek attributumainak egy egységes karakterlánc alapú kiválasztási mechanizmus által elérhetőknek kell lenniük. Ez lehetővé teszi, hogy a címkéket a metamodell pszeudo-attributumaiként, a sztereotípusokat pedig a metaosztály pszeudo-osztályaiként kezeljük, megengedve ezzel egy egyenletes átmenetet egy teljes metamodellezési képességhez (amennyiben ez szükséges).

4 ALAP CSOMAG: Adattípusok

.1 *Áttekintés*

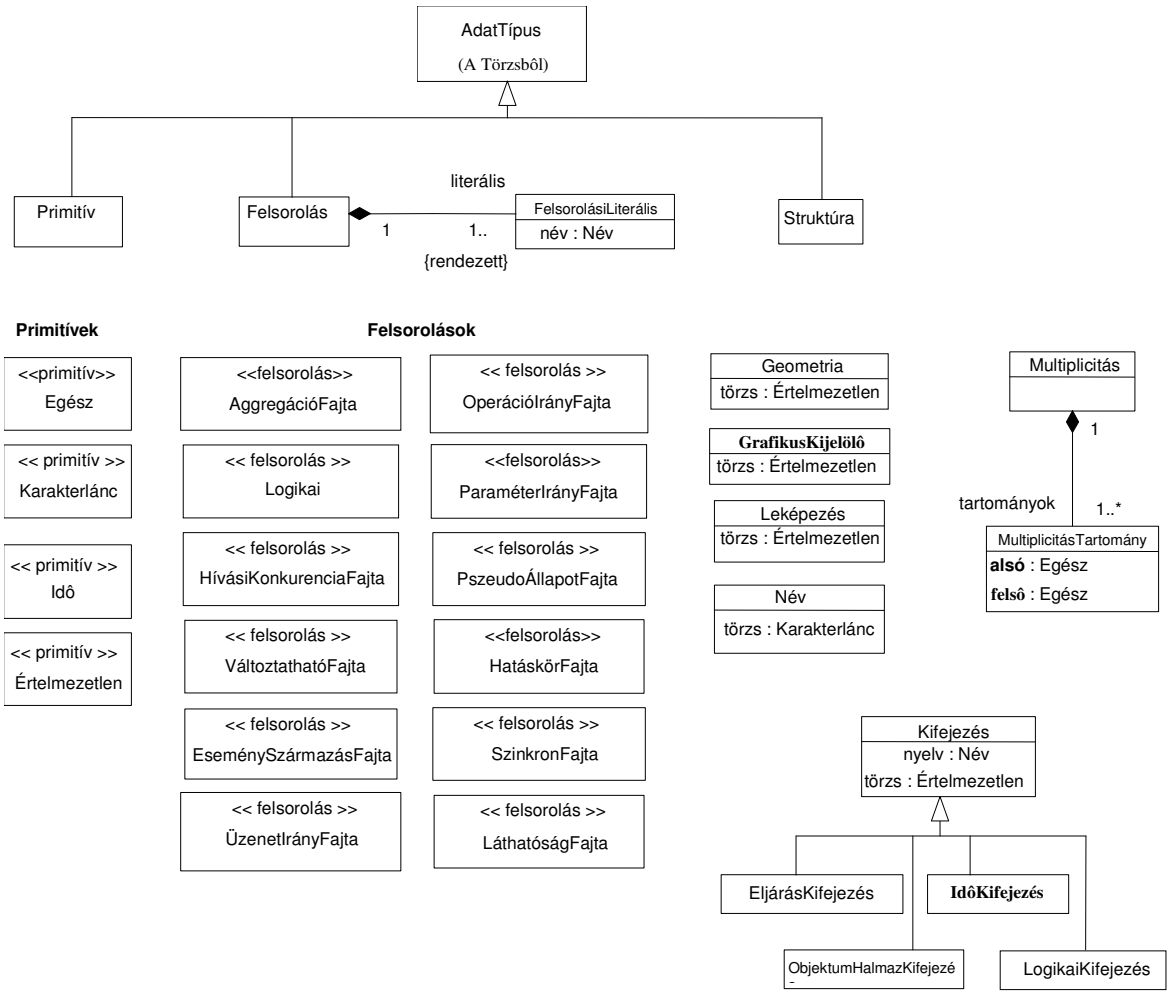
Az Adattípusok csomag az Alap csomag alcsomagja, mely specifikálja azon különböző adattípusokat, melyeket az UML használ. E szakasz struktúrája egyszerűbb, mint más

csomagoké, mivel feltételezi, hogy ezen alapfogalmak szemantikáját már jól ismeri az olvasó.

A következő szakaszok az Adattípusok csomag absztrakt szintaxisát írják le.

.2 Absztrakt szintaxis

Az Adattípusok csomag absztrakt szintaxisának grafikus leírása a 10. ábrán látható.



10. ábra: Adattípusok

A metamodellben az adattípusokat az osztály attributumok típusainak deklarálására használjuk. A diagramokban karakterlánc formában jelennek meg, nem külön "adattípus" ikonnal. Ilymódon csökkenthetők a diagramok méretei. Azonban egy bizonyos adattípus név minden előfordulása ugyanazt az adattípust jelöli.

Megjegyzendő, hogy ezek az adattípusok az UML definiálására szolgálnak, nem az UML felhasználó céljaira. Az utóbbi adattípusok a metamodellben definiált ***AdatTípus*** metaosztály példányai lesznek.

AggregációFajta (AggregationKind)

A metamodellben az ***AggregációFajta*** egy felsorolást definiál, melynek értékei *semmi (none), megosztott (shared)* és *kompozit (composite)*. Értékei azt jelentik, hogy az aggregáció mely fajtája ***Asszociáció***.

Logikai (Boolean)

A metamodellben a ***Logikai*** típus egy felsoroolást definiál, melynek értékei *hamis (false)* és *igaz (true).*

LogikaiKifejezés (BooleanExpression)

A metamodellben a ***LogikaiKifejezés*** egy programnyelvi utasítást definiál, melynek kiértékelésekor a ***Logikai*** típus egy példánya áll elő.

VáltoztathatóFajta (ChangeableKind)

A metamodellben a ***VáltoztathatóFajta*** egy felsorolást definiál, melynek értékei: *semmi (none), befagyasztott (frozen)* és *csakHozzáad (addOnly)*. Értéke azt jelenti, hogy egy ***AttributumKapocs*** vagy egy ***KapocsVégpont*** módosítható -e.

Felsorolás (Enumeration)

A metamodellben a ***Felsorolás*** az ***AdatTípus*** egy speciális válfaját definiálja, melynek értéktartománya egy definiált értékekből álló lista. Ezeket a definiált értékeket ***FelsorolásiLiterális***oknak nevezzük.

FelsorolásiLiterális (EnumerationLiteral)

Egy ***FelsorolásiLiterális*** egy atomi részt (azaz releváns alstruktúra nélkül) definiál, mely az egyenlőség vizsgálatoknál összehasonlítható.

Kifejezés (Expression)

A metamodellben egy ***Kifejezés*** olyan programnyelvi utasítást definiált, melynek a végrehajtáskori kiértékelésekor a példányok (esetleg üres) halmaza keletkezik. Egy ***Kifejezés*** nem módosítja azt a környezetet, melyben kiértékelődik.

Geometria (Geometry)

A metamodellben egy ***Geometria*** a ***Lebegőpontos*** típus egy értékhármasa, mely egy térbeli pozíciót jelent.

GrafikusKijelölő (GraphicMarker)

A metamodellben a ***GrafikusKijelölő*** a nézet elemek megjelenítésének karakterisztikáját definiálja, mint amilyen a szín, textúra, font, vonalvastagság, árnyékolás, stb.

Egész (Integer)

A metamodellben egy ***Egész*** az egészek (végtelen) halmazának (... -2, -1, 0, 1, 2 ...) egy eleme.

Leképezés (Mapping)

A metamodellben a ***Leképezés*** a ***ModellElem*** leképezéséhez használt kifejezés. A változtathatóság érdekében ***Karakterlánc*** formátumban kell ábrázolni.

ÜzenetIrányFajta (MessageDirectionKind)

A metamodellben az ***ÜzenetIrányFajta*** egy felsorolást definiál, melynek értékei: *aktiválás (activation)* és *visszatérés (return)* . Értéke az ***Üzenet*** irányát jelenti.

Multiplicitás (Multiplicity)

A metamodellben egy ***Multiplicitás*** a nemnegatív egészek egy nem üres halmazát definiálja. Olyan halmaz, mely csak nullát tartalmaz ({0}) nem vehető figyelembe egy érvényes ***Multiplicitás*** esetében. Minden ***Multitplicitás***nak legalább egy megfelelő ***Karakterlánc*** reprezentációja van.

MultiplicitásTartomány (MultiplicityRange)

A metamodellben egy ***MultiplicitásTartomány*** az egészek egy tartományát definiálja. A tartomány *felső* határa nem lehet az *alsó* határ alatt.

Név (Name)

A metamodellben egy ***Név*** olyan alapelemet definiál, melyet a ***ModellElem***ek elnevezésére, azonosítására használunk. Minden ***Név***nek van egy megfelelő ***Karakterlánc*** reprezentációja.

ObjektumHalmazKifejezés (ObjectSetExpression)

A metamodellben az ***ObjektumHalmazKifejezés*** egy programnyelvi utasítást definiál, mely a végrehajtás során példányok halmazává értékelődik ki. Az ***ObjektumHalmazKifejezések*** általánosan használatosak a célpéldányok kijelölésére egy ***Művelet***en belül.

OperációIrányFajta (OperationDirectionKind)

A metamodellben az ***OperációIrányFajta*** egy felsorolást definiál, melynek értékei: *szolgáltat (provide)* és *kér (require)*. Értékei azt jelentik, hogy egy ***Osztályozó*** kér vagy szolgáltat egy adott ***Operáció***t.

ParaméterIrányFajta (ParameterDirectionKind)

A metamodellben a ***ParaméterIrányFajta*** egy felsorolást definiál, melynek értékei: *be (in), beki (inout), ki (out)* és *visszatérés (return)* . Értéke azt fejezi ki, hogy egy paramétert egy argumentumhoz és/vagy egy visszatérési értékhez szolgáltatunk.

Primitív (Primitive)

A ***Primitív*** az egyszerű ***AdatTípus*** egy speciális fajtáját definiálja minden releváns alstruktúra nélkül.

EljárásKifejezés (ProcedureExpression)

A metamodellben az ***EljárásKifejezés*** egy programnyelvi utasítást definiál, melynek kiértékelésekor egy ***Eljárás*** példány keletkezik.

PszeudoállapotFajta (PseudostateKind)

A metamodellben a ***PszeudoállapotFajta*** egy felsorolást definiál, melynek értékei: *kezdeti (initial), teljesTörténet (deepHistory), rövidTörténet (shallowHistory), összefolyás (join), elágazás (fork), ág (branch), befejezés (final)*. Értékei egy állapotgépbeli lehetséges pszeudoállapotokat fejezik ki.

HatáskörFajta (ScopeKind)

A metamodellben a ***HatáskörFajta*** egy felsorolást definiál, melynek értékei: *osztályozó (classifier)* és *példány (instance)*. Értéke azt fejezi ki, hogy a tárolt érték a kapcsolódó ***Osztályozó*** egy példánya, vagy maga az ***Osztályozó***.

Karakterlánc (String)

A metamodellben a ***Karakterlánc*** egy folyamatos szöveget definiál.

Struktúra (Structure)

A ***Struktúra*** az ***AdatTípus*** egy speciális válfaját definiálja, mely az elnevezett részek egy rögzített számával rendelkezik.

SzinkronFajta (SynchronousKind)

A metamodellben a ***SzinkronFajta*** egy felsorolást definiál, melynek értékei: *szinkron (synchronous)* és *aszinkron (asynchronous)*. Értéke azt fejezi ki, hogy végrehajtáskor egy ***HívásiMűvelet*** során az ***Üzenet*** mely fajtája álljon elő.

Idő (Time)

A metamodellben az ***Idő*** egy olyan értéket definiál, mely egy abszolút vagy relatív idő és térbeli pillanatot reprezentál. Az ***Idő*** megfelelő karakterlánc reprezentációval rendelkezik.

IdőKifejezés (TimeExpression)

A metamodellben az ***IdőKifejezés*** egy olyan programnyelvi utasítást definiál, melynek kiértékelésekor egy ***Idő*** példány keletkezik.

Értelmezetlen (Uninterpreted)

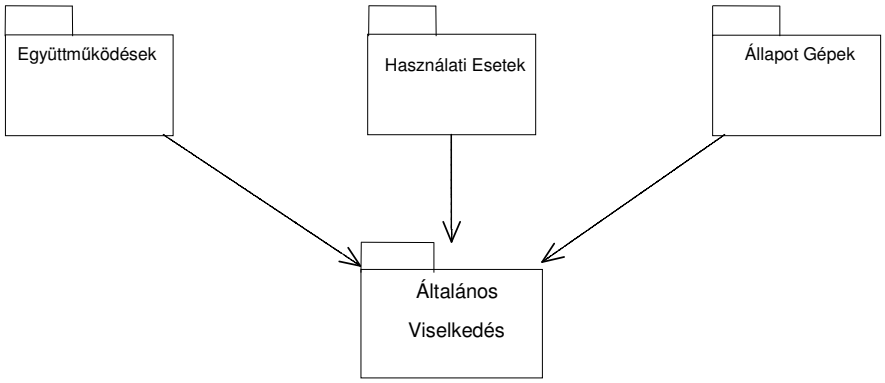
A metamodellben az ***Értelmezetlen*** egy blob, jelentése domén specifikus, éppenezért nincs definíciója az UML -ben.

LáthatóságFajta (VisibilityKind)

A metamodellben a ***LáthatóságFajta*** egy felsorolást definiál, melynek értékei: *publikus (public), védett (protected)* és *privát (private)*. Értéke kifejezi, hogy az elem, melyre utal, látható -e a névterületen kívülről.

4. III. RÉSZ: VISELKEDÉSI ELEMEK

A harmadik rész definiálja a szuperstruktúrát az UML -beli viselkedés modellezéshez, a Viselkedési Elemek csomaghoz. A Viselkedési Elemek csomag négy alsóbb szintű csomagból áll: Általános Viselkedés, Együttműködések, Használati Esetek és Állapotgépek. Az Általános Viselkedés meghatározza a viselkedési elemekhez szükséges alapfogalmakat. Az Együttműködések csomag egy viselkedési kontextust specifikál azon modell elemek számára, melyek által egy meghatározott feladat lebonyolítható. A Használati Esetek csomag viselkedést specifikál szereplők (aktorok) és használati esetek alkalmazásával. Az Állapotgépek csomag véges állapotú átmeneti rendszerek felhasználásával definiál viselkedést.



11. ábra: A Viselkedési Elemek Csomagjai

A HARMADIK RÉSZ TARTALMA

1 Általános viselkedés

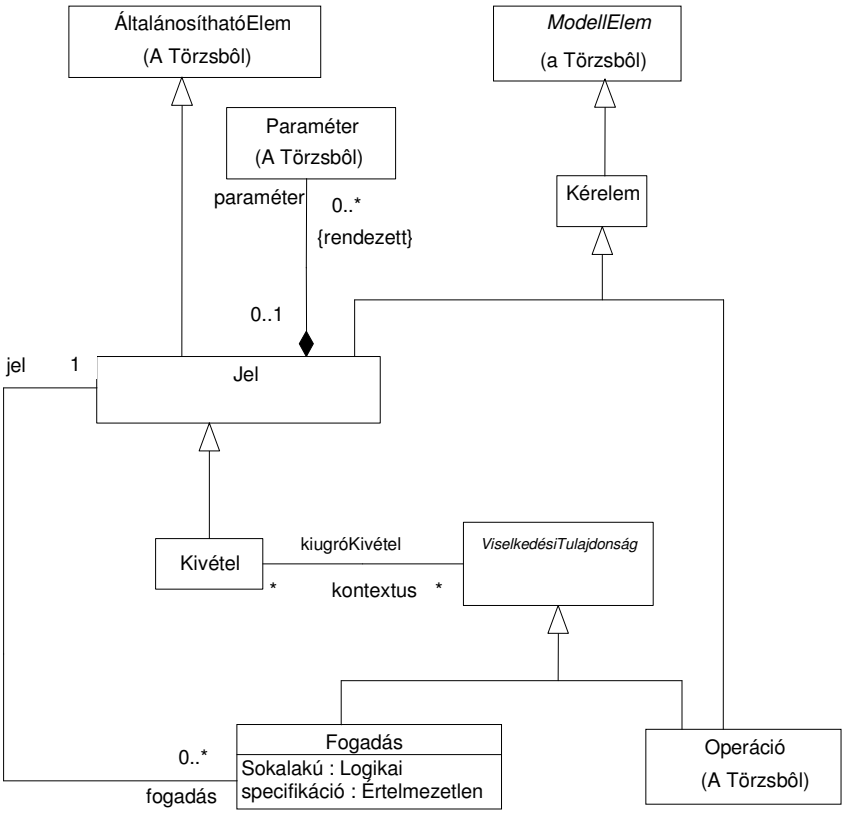
.1 Áttekintés

Az általános viselkedés csomag a legalapvetőbb a Viselkedési Elemek alcsomagjai közül. Meghatározza azokat az alapfogalmakat, melyek a dinamikus elemekhez szükségesek, valamint infrastruktúrát biztosít az Együttműködések, Állapot Gépek és Használati Esetek támogatásához.

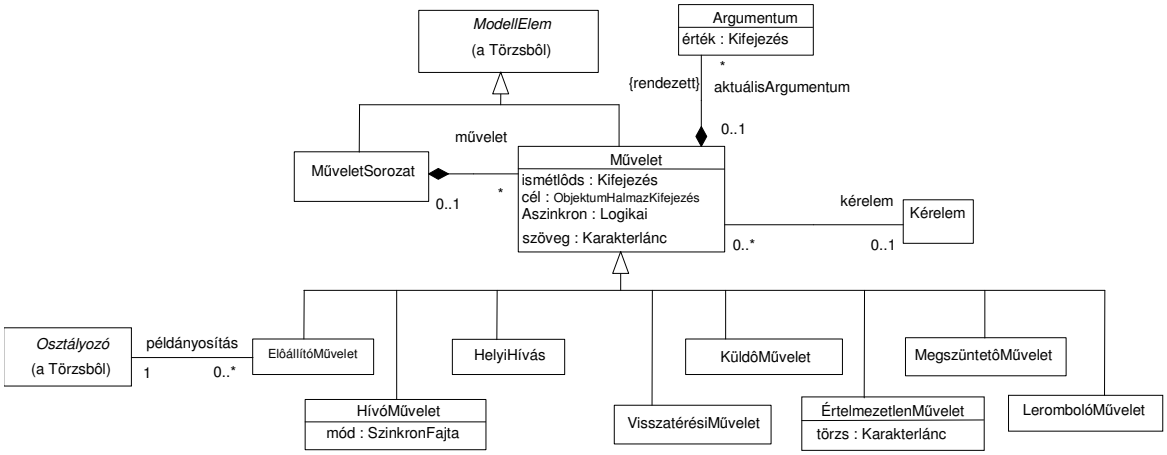
A következő szakaszok az Általános Viselkedés csomag absztrakt szintaxisát, jól képzett szabályait és szemantikáját írják le.

.2 *Absztrakt szintaxis*

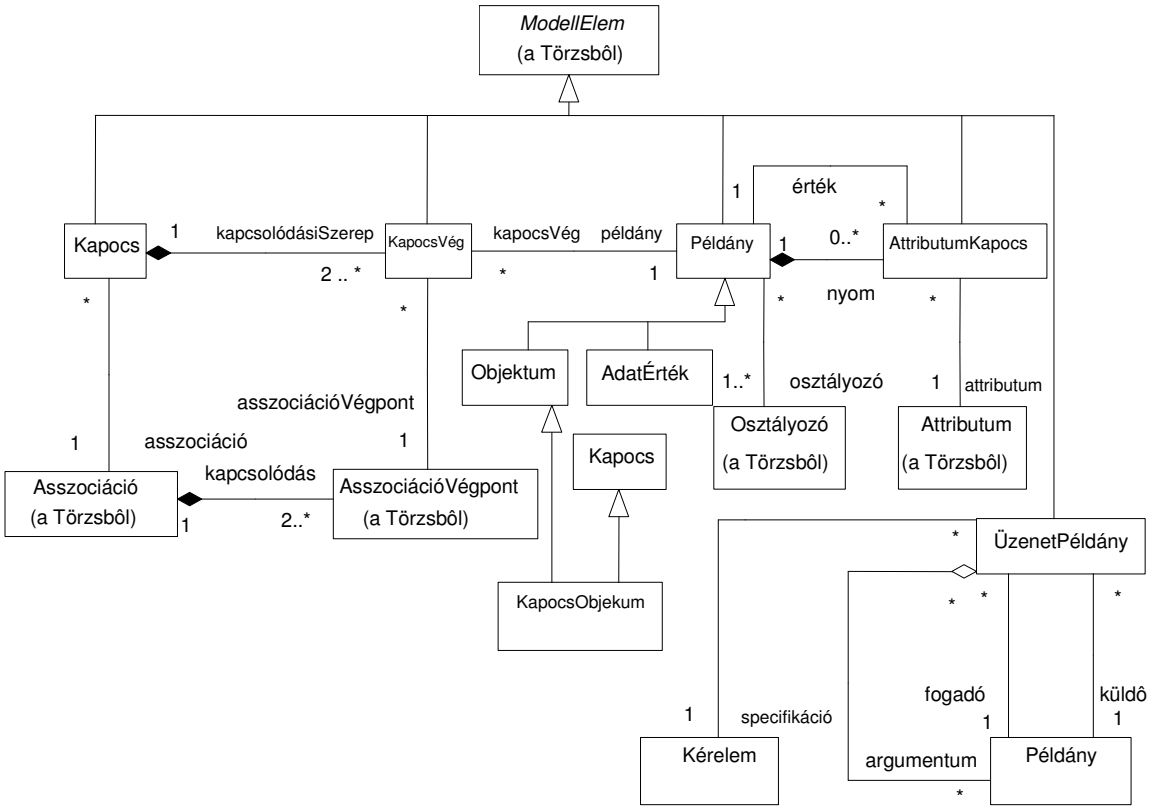
Az Általános Viselkedés csomag absztrakt szintaxisa grafikus formába látható a 12., 13. és 14. ábrán. A 12. ábra szemlélteti azokat a modell elemeket, melyek a ***Kérelmeket*** definiálják. Ide tartoznak a ***Jelek*** és az ***Operációk***. A 13. ábra azon modell elemeket illusztrálja, melyek a különböző műveleteket határozzák meg, mint amilyen az ***ElőállításiMűvelet, HívásiMűvelet*** és ***KüldésiMűvelet***. A 14. ábra olyan modell elemeket jelenít meg, melyek ***Példányokat*** és ***Kapcsokat*** definiálnak.



12. ábra: Általános Viselkedés - Kérelmek



13. ábra: Általános Viselkedés - Műveletek



14. ábra: Általános Viselkedés - Példányok és Kapcsok

A következő metaosztályokat tartalmazza az Általános Viselkedés csomag:

Művelet (Action)

A *művelet* egy végrehajtható programnyelvi utasítás specifikációja, mely egy számítási eljárás absztrakcióját képezi. E számítási eljárás a modell állapotának megváltozását eredményezi, megvalósulása egy objektum felé történő üzenetküldéssel, vagy egy attributum értékének megváltoztatása által jön létre.

A metamodellben egy ***Művelet*** a ***MűveletSorozat*** része, és tartalmazhatja a *cél* specifikációját, csakúgy mint az elküldött ***Kérelem*** *argumentumainak* (aktuális paramétereinek) specifikációját.

A *cél* típus az ***ObjektumHalmazKifejezés*** metaattributuma, mely végrehajtásakor nullát, vagy meghatározottabb ***Példányokat*** ad eredményül, melyek az elküldött ***Kérelem*** megcélzott címzettei. Hasonlóan kapcsolódik az ***Argumentumok*** listájával, mely futási idő alatt értékelődik ki a ***Kérelem*** aktuális argumentumaivá. Az *ismétlődés* metaattributum meghatározza, hogy a művelet végrehajtásakor a ***Kérelem*** hányszor legyen elküldve.

A ***Művelet*** egy absztrakt metaosztály.

Attributumok (Attributes)

ismétlődés (recurrence) Egy ***Kifejezés***, mely megadja, hogy a ***Műveletet*** hányszor kell végrehajtani.

cél (target) Egy ***ObjektumHalmazKifejezés***, mely meghatározza a ***Művelet*** célját.

Asszociációk (Associations)

kérelem (request) A ***Kérelem Művelet*** által elküldött specifikációja.

aktuálisArgumentum (actualArgument) ***Kifejezések*** sorozata, melyek meghatározzák a ***Művelet*** kiértékelésekor igényelt aktuális argumentumokat.

MűveletSorozat (ActionSequence)

A *művelet sorozat* műveletek csoportja.

A metamodellben a ***MűveletSorozat*** a ***Műveletek*** aggregációja. A birtokolt ***Állapot*** vagy ***Átmenet*** viselkedését írja le.

Asszociációk (Associations)

művelet (action) ***Művelet***sorozat szekvenciális végrehajtása. Az egyes műveletek atomi egységként értelmezettek.

Argumentum (Argument)

Az *argumentum* azokat az aktuális értékeket reprezentálja, melyek egy elküldött kérelemhez adódnak át és a művelet során aggregálódnak.

A metamodellben egy ***Argumentum*** a ***Művelet*** része és a ***Kifejezés*** típusának metaattributumát és *értékét* tartalmazza.

Attributumok (Attributes)

érték (value) Egy ***Kifejezés***, mely kiértékelésekor meghatározza az aktuális ***Példányt***.

AttributumKapocs (AttributeLink)

Egy *attributum kapocs* egy névvel ellátott hely egy példányon belül, mely egy attributum értékét tárolja.

A metamodellben az ***AttributumKapocs*** a ***Példány*** egyik állapota és egy ***Attributum*** értékét tárolja.

Asszociációk (Associations)

érték (value) Az a ***Példány***, mely az ***AttributumKapocs*** értékét képviseli.

attributum (attribute) Az az ***Attributum***, melyből az ***AttributumKapocs*** származik.

HívásiMűvelet (CallAction)

A *hívási művelet* olyan művelet, melyet egy operáció meghívása eredményez. Egy hívási művelet lehet szinkron, vagy aszinkron. Ez azt jelenti, hogy az operáció hívása szinkron vagy aszinkron módon történik.

A metamodellben a ***HívásiMűvelet*** a ***Művelet*** altípusa. A kijelölt példány, vagy példányok specifikálása a *cél* kifejezésen keresztül történik. Az aktuális argumentumok kijelölése azon az *argumentum* asszociáción keresztül történik, mely a ***Művelet***től öröklődik. Az eredményül kapott operációt az elküldött ***Kérelem*** specifikálja, mely ebben az esetben egy ***Operáció***.

Attributumok (Attributes)

mód (mode) Egy felsorolás, mely megadja, hogy az elküldött ***Operáció*** szinkron vagy aszinkron lesz.

szinkron Jelzi, hogy a hívó várakozik az ***Operáció*** végrehajtásának befejezésére.

aszinkron Jelzi, hogy a hívó *nem* várakozik az ***Operáció*** végrehajtásának befejezésére, hanem azonnal folytatja tevékenységét.

ElőállítóMűvelet (CreateAction)

Az *előállító művelet* olyan művelet, ami valamelyik osztályozó egy példányának létrehozását eredményezi.

A metamodellben az ***ElőállítóMűvelet*** a ***Művelet*** altípusa. Az ***Osztályozó*** osztályt az ***ElőállítóMűvelet*** *példányosítás* asszociációja jelöli ki.

Asszociációk (Associations)

osztályozó (classifier) Az az ***Osztályozó***, melynek ***Példány***a létrejön az ***ElőállítóMűvelet*** végrehajtásakor.

LeromlóMűvelet (DestroyAction)

A *leromboló művelet* az általa meghatározott objektum lerombolását eredményezi.

A metamodellben a ***LerombolóMűvelet*** a ***Művelet*** alosztálya. A kijelölt objektumot a ***Művelet*** *cél* asszociációja specifikálja.

AdatÉrték (DataValue)

Az *adat érték* egy azonosítatlan példány.

A metamodellben a ***AdatÉrték*** a ***Példány*** alosztálya, mely nem képes megváltoztatni állapotát, azaz az összes hozzá alkalmazható ***Operáció*** egyszerű függvény vagy lekérdezés. Az ***AdatÉrrék***eket jellegzetesen attributum értékekként használjuk.

Kivétel (Exception)

A *kivétel* egy jel, melyet a viselkedési tulajdonság generál, jellegzetesen a végrehajtási hibák esetén. A metamodellben a ***Kivétel*** a ***Jel***től származik. A ***Kivétel*** a ***ViselkedésiTulajdonsgág***gal áll kapcsolatban, mely őt generálta.

Attributumok (Attributes)

törzs (body) A ***Kivétel*** leírásának formátuma nem definiált az UML -en belül.

Asszociációk (Associations)

viselkedésiTulajdonság (behavioralFeature) Azon ***ViselkedésiTulajdonság***ok halmaza, melyek a kivételt generálták.

Példány (Instance)

A *példány* konstrukció egy olyan egyedet definiál, melyhez operációk csoportja alkalmazható, és mely állapottal rendelkezik. Ez az állapot tárolja az operációk hatásait.

A metamodellben a ***Példány*** legalább egy ***Osztályozóval*** áll kapcsolatban, mely deklarálja a struktúráját és a viselkedését. Attributum értékekkel rendelkezik és ***Kapcsok*** csoportjához kapcsolódik. Az attributum értékek és a ***Kapcsok*** halmaza egyaránt illeszkedik az ***Osztályozójának*** definícióival. A két halmaz implementálja a ***Példány*** aktuális állapotát. A ***Példány*** egy absztrakt metaosztály.

Asszociációk (Associations)

attributumKapocs (attributeLink) ***AttributumKapcsok*** halmaza, mely a ***Példány*** attributum értékeit tárolja.

kapocsVégpont (linkEnd) Az illesztett ***Kapcsok KapocsVégpontjainak*** halmaza, mely kapcsok a ***Példányhoz*** kötődnek.

osztályozó (classifier) Azon ***Osztályozók*** halmaza, melyek deklarálják a ***Példány*** struktúráját.

Kapocs (Link)

A *kapocs* szerkezet összeköttetést képez a példányok között.

A metamodellben a ***Kapocs*** az ***Asszociáció*** példánya. ***KapocsVégpontok*** csoportjával rendelkezik, mely illeszkedik az ***Asszociáció AsszociációVégpont*** halmazával. Egy ***Kapocs*** összeköttetést definiál a ***Példányok*** között.

Asszociációk (Associations)

asszociáció (association) Az ***Asszociáció*** a ***Kapocs*** deklarációja.

kapocsSzerep (linkRole) ***KapocsVégpontok*** sorozata, mely a ***Kapcsot*** alkotja.

KapocsVégpont (LinkEnd)

A *kapocs végpont* - mint ahogy a neve is mutatja - a kapocs egyik végpontja.

A metamodellben a ***KapocsVégpont*** a ***Kapocs*** egy része, mely egy ***Példányhoz*** kapcsolódik. Megfelel a ***Kapocs Asszociáció*** egy ***AsszociációVégpontjának***.

Asszociációk (Associations)

példány (instance) Az a ***Példány,*** mely a ***KapocsVégponthoz*** kötődik.

asszociációVégpont (associationEnd) Az az ***AsszociációVégpont***, mely a ***KapocsVégpont*** deklarációja.

KapocsObjektum (LinkObject)

A *kapocs objektum* egy saját attributumérték halmazzal rendelkező kapocs, melyhez operációk halmaza alkalmazható.

A metamodellben a ***KapocsObjektum*** a ***Példányok*** csoportja közötti kapcsolatot testesíti meg, ahol magának a kapcsolatnak attributumérték-halmaza lehet és melyhez ***Operációk*** halmaza alkalmazható. Az ***Objektum*** és a ***Kapocs*** alosztálya.

HelyiHívás (LocalInvocation)

A *helyi hívás* a művelet speciális típusa, mely egy *helyi* operációt hív meg (egy operációt "önmagán"). A hívás e típusa az állapot gép közvetítése nélkül foglal helyet;

azaz nem generál hívási eseményt. Egy objektum helyi segédszolgáltatást nyújtó eljárásának meghívása jó példa a ***HelyiHívásra***. Ellentétként megemlíthető, hogy egy "önmagán" történő ***HívásiMűvelet*** mindig egy egy eseményt eredményez.

A metamodellben a ***HelyiHívás*** az ***Operációval*** áll kapcsolatban, mely a ***Kérelemmel*** fennálló kapcsolatán keresztül hívja meg. Az *argumentum* asszociáció meghatározza az ***Operáció*** argumentumait, melyeket az *argumentum* asszociáció specifikál. (a ***Művelettől*** öröklődik).

ÜzenetPéldány (MessageInstance)

Az *üzenet példány* kommunikációk testesít meg két példány között.

A metamodellben az ***ÜzenetPéldány*** a ***Kérelem*** alosztályának egy példánya, hasonló a ***Jelhez*** és a ***Kérelemhez***. Küldővel és fogadóval rendelkezik, és argumentum halmaza is lehet, mely argumentumok mind ***Példányok***.

Asszociációk (Associations)

specifikáció (specification) A ***Kérelem***, melytől az ***ÜzenetPéldány*** származik.

küldő (sender) Az a ***Példány***, mely az ***ÜzenetPéldányt*** küldte.

fogadó (receiver) Az a ***Példány***, mely fogadja az ***ÜzenetPéldányt***.

argumentumok (arguments) ***Példányok*** sorozata, melyek az ***ÜzenetPéldány*** argumentumai.

Objektum (Object)

Az *objektum* egy példány, mely egy osztálytól származik.

A metamodellben az ***Objektum*** a ***Példány*** alosztálya és legalább egy ***Osztálytól*** származik. Az ***Osztályok*** halmaza dinamikusan változhat, ami azt jelenti, hogy az ***Objektum*** tulajdonságainak halmaza változik az élettartama alatt.

Vétel (Reception)

A *vétel* egy deklaráció, megadja, hogy egy osztályozó elő van készítve egy jel vételéhez. A vétel kijelöl egy jelet és meghatározza a várt viselkedési választ. Egy vétel a várt viselkedések összesítése; a jel kezelésének részleteit az állapot gép specifikálja.

A metamodellben a ***Vétel*** a ***ViselkedésiTulajdonság*** egy alosztálya és deklarálja, hogy az ***Osztályozó***, mely tartalmazza a tulajdonságot, visszahat arra a jelre, melyet a vételi tulajdonság jelöl ki. A *Sokalakú (isPolymorphic)* attributum meghatározza, hogy a viselkedés sokalakú, vagy nem; az *igaz* érték jelzi, hogy a viselkedés nem mindig ugyanaz és hatással lehet rá az állapot. A specifikáció feltünteti a jelre várt választ is.

Attributumok (Attributes)

Sokalakú (isPolymorphic) Rögzített -e a válasz a *Jelre*. Ha igaz, akkor a válasz függhet az *Osztályozó* állapotától és az alosztályok által felülíródhat.

Amennyiben hamis, akkor a jelre adott válasz mindig ugyanaz, tekintet nélkül az *Osztályozó* állapotától, és nem írhatják felül az alosztályok.

specifikáció (specification) Azon osztályozó hatásainak ***Kifejezésként*** megadott leírása, mely egy jelet fogad.

Asszociációk (Associations)

jel (signal) A ***Jel*** az amit az ***Osztályozó*** a kezeléshez előkészít.

Kérelem (Request)

A *kérelem* azon stimulus (inger) specifikációja, melyet a példányokhoz küldünk. Vagy operáció, vagy jel lehet.

A metamodellben a ***Kérelem*** a ***ViselkedésiTulajdonság*** egy absztrakt alosztálya.

VisszatérésiMűvelet (ReturnAction)

A *visszatérési művelet* , olyan művelet, mely egy értéknek a hívóhoz való visszaadását eredményezi.

A metamodellben a ***VisszatérésiMűvelet*** értékek reprezentációja olyan argumentumokként valósul meg, melyek egy ***Műveletttől*** öröklődnek.

KüldőMűvelet (SendAction)

A *küldő művelet* olyan művelet, mely egy jel (aszinkron) küldését eredményezi. A jel az ObjektumHalmazKifejezésen keresztül a fogadók halmazához irányítható, vagy implicit módon küldhető meghatározatlan fogadókhoz, melyeket külső mechanizmusok definiálnak. Például, ha a jel egy kivétel, akkor a fogadót az alárendelt futásidejű rendszer mechanizmusok határozzák meg.

A metamodellben a ***KüldőMűvelet*** azon *kérelem* asszociáció által kapcsolódik a ***Jellel***, mely a ***Művele***től öröklődik. Az aktuális argumentumokat az *argumentum* asszociáció határozza meg, mely szintén a ***Művele***től öröklődik.

Jel (Signal)

A *jel* a példányok közötti kommunikációhoz használt aszinkron stimulus (inger) specifikációja. A fogadó példány az állapot gép által kezeli a jelet. A jel általánosítható elem és a jelet kezelő osztályoktól függetlenül van definiálva. A fogadás annak deklarációja, hogy egy osztály hogyan kezel egy jelet, de a tényleges kezelést az állapotgép specifikálja.

A metamodellben a ***Jel*** a ***Kérelem*** alosztálya, melyet a ***KüldőMűvelet*** küldött. ***ÁltalánosíthatóElem***, és ***Paraméter***halmazt gyűjt össze. Egy ***Jel*** mindig aszinkron.

Asszociációk (Associations)

fogadás (reception) ***Fogadás***ok halmaza, mely jelzi, hogy az *Osztályok* elő vannak készítve a jel kezelésére.

MegszüntetőMűvelet (TerminateAction)

A *megszüntető művelet* egy objektum "önpusztítását" eredményezi.

A metamodellben a ***MegszüntetőMűvelet*** a ***Művelet*** alosztálya.

ÉrtelmezetlenMűvelet (UninterpretedAction)

Az *értelmezetlen művelet* az összes olyan műveletet reprezentálja, melyek nincsenek explicit módon megtestesítve az UML -ben.

A határeseteket tekintve bármely művelet valmely példányra vonatkozó hívás vagy létrehozás (pl. Smalltalk). Azonban a praktikusabb kifejezésekben, műveletekben, mint amilyenek a hozzárendelések és a feltételes utasítások, megadhatók értelmezetlen műveletekként is, csakúgy mint bármely más nyelv meghatározott műveletei, melyek sem hívási, sem küldési műveletek.

Attributumok (Attributes)

törzs (body) A művelet definíciója.

.3 Jól képzett szabályok

A következő jól képzett szabályokat alkalmazzuk az Általános Viselkedés csomaghoz.

AttributumKapocs (AttributeLink)

[1] A ***Példány*** típusának illeszkednie kell az ***Attributum*** típusával.

```
self.value.classifier->includes(self.attribute.type)
```

HívóMűvelet (CallAction)

[1] A művelethez megadott argumentumok típusainak és sorrendjének illeszkednie kell a ***Kérelem*** paramétereivel.

```
(self.actualArgument->size > 0)
   implies (Sequence{1..self.actualArguments->size})->
      forAll (x |
         self.actualArgument->at(x).type =
         self.message.parameter->at(x).type)
Note: parameter referes to Signal or Operation (downcast)
```

[2] Egy ***HívóMűveletnek*** pontosan egy céljának kell lennie.

```
self.target->size = 1
```

[3] Az elküldött ***Kérelem*** típusának ***Operáció***nak kell lennie.

```
self.message->notEmpty
and
self.message.oclIsTypeOf(Operation)
```

ElőállítóMűvelet (CreateAction)

[1] Egy ***ElőállítóMűvelet***nek nem lehet cél kifejezése.

```
self.target->isEmpty
```

LerombolóMűvelet (DestroyAction)

[1] Egy ***LerombolóMűveletnek*** nem lehetnek argumentumai

```
self.actualArgument->size = 0
```

AdatÉrték (DataValue)

[1] Egy ***AdatÉrték*** pontosan egy ***Osztályozó***tól származik, mely egy ***AdatTípus***.

```
(self.classifier->size = 1)
and
self.classifier.oclIsKindOf(DataType)
```

[2] Egy ***AdatÉrték*** nem rendelkezik ***AttributumKapcsokkal***.

```
self.slot->isEmpty
```

Példány (Instance)

[1] Az ***AttributumKapcsok*** illeszkednek az ***Osztályozó***kon belüli deklarációkkal.

```
self.slot->forAll ( al |
   self.classifier->exists ( c |
      c.allAttributes->includes ( al.attribute ) ) )
```

[2] A ***Kapcsok*** illeszkednek az ***Osztályozó***kon belüli deklarációkkal.

```
self.allLinks->forAll ( l |
   self.classifier->exists ( c |
      c.allAssociations->includes ( l.association ) ) )
```

[3] Ha két ***Operáció***nak ugyanaz a szignatúrája, akkor azonosaknak kell lenniük.

```
self.classifier->forAll ( c1, c2 |
   c1.allOperations->forAll ( op1 |
      c2.allOperations->forAll ( op2 |
         op1.hasSameSignature (op2)  implies op1 = op2 ) ) )
```

[4] Nincsenek névütközések az ***AttributumKapcsok*** és az ellenoldali ***KapocsVégpontok*** között.

```
self.slot->forAll( al |
   not self.allOppositeLinkEnds->exists( le | le.name = al.name ) )
and
self.allOppositeLinkEnds->forAll( le |
   not self.slot->exists( al | le.name = al.name ) )
```

Kiegészítő operációk

[1] Az *összesKapocs (allLinks)* operáció egy olyan halmazt eredményez, mely magának a ***Példánynak*** az összes ***Kapcsát*** tartalmazza.

```
allLinks : set(Link);
allLinks = self.linkEnd->collect ( l | l.link )
```

[2] Az *összesEllenoldaliKapocsVégpont (allOppositeLinkEnds)* operáció egy olyan halmazt eredményez, mely azon ***Kapcsok*** összes ***KapocsVégpontját*** tartalmazza, melyek egy másik ***KapocsVégpont*** által kapcsolódnak a ***Példányhoz***.

```
allOppositeLinkEnds : set(Link);
allOppositeLinkEnds = self.allLinks->collect ( l |
   l.linkRole )->select ( le | le.instance <> self )
```

Kapocs (Link)

[1] A ***KapocsVégpontok*** halmazának illeszkednie kell az ***Asszociáció AsszociációVégpont***jaiból álló halmazzal.

```
Sequence {1..self.linkRole->size}->forAll ( i |
 self.linkRole->at (i).associationEnd = self.association.connection->at (i) )
```

[2] Ugyanannak az ***Asszociáció***nak nincs két ***Kapcsa***, melyek ugyanolyan módon létesítenének kapcsolatot ugyanazon ***Példány***halmaz tagjai között.

```
self.association.instance->forAll ( l |
  Sequence {1..self.linkRole->size}->forAll ( i |
     self.linkRole.instance = l.linkRole.instance ) implies self = l )
```

KapocsVégpont (LinkEnd)

[1] A ***Példány*** típusának egyeznie kell az ***AsszociációVégpont*** típusával.

```
self.instance.classifier->includes (self.associationEnd.type)
```

KapocsObjektum (LinkObject)

[1] Az ***Osztályozó***k egyikének ugyanannak kell lennie, mint az ***Asszociáció***.

```
self.classifier->includes(self.association)
```

[2] Az ***Asszociáció***nak az ***AsszociációOsztály*** egy fajtájának kell lennie.

```
self.association.oclIsKindOf(AssociationClass)
```

ÜzenetPéldány (MessageInstance)

[1] Az argumentumok típusának illeszkednie kell a ***Kérelem*** paramétereinek típusával.

```
self.argument->size = self.specification.parameter->size
and
Sequence {1..self.argument->size}->forAll ( i |
   self.argument->at (i).classifier->includes (
      self.specification.parameter->at (i).type ) )
```

```
-- Note: parameter refers to the parameter of the operation or signal
-- subclasses of request.
```

Objektum (Object)

[1] Az ***Osztályozó***k mindegyikének az ***Osztály*** egy fajtájának kell lennie.

```
self.classifier->forAll ( c | c.oclIsKindOf(Class))
```

Jel (Signal)

[1] Egy ***Jel*** mindig aszinkron és mindig egy hívás.

```
self.isAsynchronous and self.direction = activation
```

Fogadás (Reception)

[1] A ***Fogadás*** nem lehet lekérdezés.

```
not self.isQuery
```

Kérelem (Request)

Kiegészítő operációk

[1] Egy ***Kérelem*** *paramétere* a ***Jel*** vagy az Operáció paramétere.

[1] The *parameter* of a ***Request*** is the parameter of the ***Signal*** or Operation.

```
parameter  : set(Parameter);
parameter = if self.oclIsKindOf(Operation)
            then self.oclAsType(Operation).parameter
            else if self.oclIsKindOf(Signal)
                  then self.oclAsType(Signal).parameter
                  else Set {}
            endif endif
```

KüldőMűvelet (SendAction)

[1] Az aktuális argumentumok típusainak és sorrendjének illeszkednie kell a ***Kérelem (Jel*** vagy ***Operáció)*** paramétereivel.

```
(self.actualArgument->size > 0)
  implies (Sequence{1..self.actualArgument->size}->
     forAll (x |
        self.actualArgument->at(x).type =
        self.message.parameters->at(x).type))
-- note: parameters apply to signal or operation (downcast)
```

[2] Az elküldött ***Kérelem*** típusa egy ***Jel***.

```
self.message->notEmpty
and
self.message.oclIsKindOf (Signal)
```

[3] Egy ***Kivétel*** céljának üresnek (implicit) kell lennie.

```
self.message.oclIsKindOf(Exception) implies (self.target = NULL)
```

MegszüntetőMűvelet (TerminateAction)

[1] Egy ***MegszüntetőMűvelet***nek nem lehetnek argumentumai.

```
self.actualArgument->size = 0
```

.4 Szemantika

Ez a szakasz az Általános Viselkedés csomag elemeinek szemantikáját írja le.

Objektum és AdatÉrték (Object and DataValue)

Az *objektum* egy példány, mely egy osztályból származik - struktúrája és viselkedése megfelel az osztályáénak. Minden objektum ugyanabból az osztályból származik, ugyanúgy épülnek fel, de mindegyiküknek saját attributum kapocs halmaza van. Minden attributum kapocs egy példányra, általában egy adatértékre utal. Az attributum kapcsok száma ugyanazon névvel eleget tesz az osztályban lévő megfelelő attributum multiplicitásának. A halmaz módosulhat a megfelelő attributumbeli specifikációnak megfelelően; azaz minden hivatkozott példánynak az attributum egy (al)típusából kell származnia és attributum kapcsok adhatók hozzá, vagy vehetők el az attributum változtatható tulajdonságának megfelelően.

Egy objektumnak több osztálya is lehet, azaz, több osztályból is származhat. Ebben az esetben az objektum birtokolni fogja ezen összes osztályban deklarált tulajdonságokat, beleértve a struktúrális és a viselkedési tulajdonságokat. Továbbá az osztályok halmaza (azaz a tulajdonságoknak azon halmaza, melyhez az objektum idomul) az idő során változhat. Új osztályok adódhatnak az objektumhoz, illetve régiek kapcsolódhatnak le. Ez azt jelenti, hogy az új osztályok tulajdonságai dinamikusan adódnak az objektumhoz, és a tulajdonságok abban az osztályban vannak deklarálva, mely dinamikusan lett eltávolítva az objektumtól. Az attritubum kapcsok és az ellenoldali kapocs végpontok közötti névütközések engedélyezettek és minden operációnak, mely alkalmazható az objektumhoz, egyedi szignatúrával kell rendelkeznie.

A példány egy másik fajtája az *adat érték*, mely egy azonosítatlan példány. Továbbá egy adatérték nem változtathatja meg az állapotát - az összes operáció, mely egy adatértékhez alkalmazható, lekérdezés és nem okozhat semmilyen mellékhatást. Mivel nem lehetséges különbséget tenni két olyan adattípus között, melyek azonosak, inkább filozófiai kérdés annak eldöntése, hogy engedélyezett legyen -e több adatérték reprezentációja ugyanazzal az értékkel, vagy mindegyik értéknek csakis egy adatérték legyen megfeleltetve. Továbbá egy adatérték nem képes megváltoztatni típusát.

Kapocs (Link)

A *kapocs* egy kapcsolat a példányok között. Minden kapocs egy asszociáció példány, azaz a kapocs a kapcsolódó oszályozó példányait (alosztályait) köti össze. A példány kontextusban egy *ellenoldali végpont* definiálja a példányok azon halmazát, mely példányok ugyanazon asszociáció kapcsain keresztül csatlakoznak más példányokhoz. Ezen kívül minden példány azon kapocs végponton keresztül csatlakokzik a kapcsához, mely kapocs végpont ugyanabból az asszociáció végpontból származik. Azonban egy meghatározott ellenoldali végpont felhasználhatóságának érdekében annak a kapocs végpontnak, mely a példányhoz kapcsolódik, navigálhatónak kell lennie. Egy példány felhasználhatja ellenoldali végpontjait a kapcsolódó példányok eléréséhez. Egy példány képes kommunikálni az ellenoldali végpontok példányaival és hivatkozásokat használhat hozzájuk, mint argumentumokat, vagy válasz értékeket a kommunikációkban.

A *kapocs objektum* a kapocs egy speciális fajtája, ugyanakkor egyben egy objektum is. Mivel egy objektum megváltoztathatja az osztályokat, ez igaz egy kapocs objektumra is.

Kérelem, Jel, Kivétel és ÜzenetPéldány

A *kérelem* a példányok közötti kommunikáció specifikációja. E specifikáció egy olyan példány eredménye, mely meghatározott fajtájú műveleteket hajt végre: hívó művelet, létrehozó művelet, leromboló művelet és visszatérési művelet.

A kérelemnek két fajtája létezik: *jel* és *operáció*. Az előbbit egy reakció triggerelésére (kiváltására, aktiválására) használjuk a fogadóban aszinkron módon és válasz nélkül, az utóbbi egy operáció specifikációja, mely lehet szinkron vagy aszinkron és választ kérhet a fogadótól a küldőhöz. Amikor egy példány más példánnyal kommunikál, egy *üzenet példány* kerül elküldésre a két példány között. Ennek az *üzenet példánynak* van egy küldője, egy fogadója és rendelkezhet argumentumhalmazzal is a kérelem specifikációjának megfelelően. Egy jel csatlakozhat egy osztályozóhoz, ami azt jelenti, hogy az osztályozó példányai képesek lesznek a jel vételére. Ezt az osztályozó által deklarált *fogadás* segíti elő.

A *kivétel* a jel egy speciális fajtája, rendszerint jelhiba állapotban alkalmazzuk. A kivétel küldője megszakítja a végrehajtást és az végrehajtás újrakezdődik a végrehajtás vevőjével, mely lehet a küldő önmaga is. Más jelektől eltérően a fogadót implicit módon meghatározza a kölcsönhatás sorozat a végrehajtás során és nincs explicit módon specifikálva.

Az üzenet példány fogadása egy hívási művelettől származik egy olyan példány által, mely a fogadón végrehajtott operáció meghívását idézi elő. A fogadó végrehajtja azt a metódust, mely azon osztály teljes deszkriptorában található, mely megfelel az operációnak. Egy jel fogadása egy példány által egy átmenetet és későbbi hatásokat idézhet elő, ahogy azt az állapot gép specifikálta a címzett osztályozója számára. A viselkedés e formájának leírása az Állapot Gépek csomagban található. Megjegyzendő, hogy a meghívott viselkedést a metódusok és az állapotgép átmenetek írják le. Az *Operációk* és a *Fogadások* csupán azt deklarálják, hogy egy osztályozó elfogad egy adott *Kérelmet*, de nem specifikálják az implementációt.

Művelet (Action)

A *művelet* egy kiszámítható utasítást specifikál. Az üzenet minden fajtája a művelet alosztályaként van definiálva. Az UML -ben a következő fajta műveletek vannak definiálva:

- a *küldő művelet (send action)* egy olyan művelet, melyben egy üzenet példány generálódik. Ez az üzenet példány egy jel eseményt idéz elő a fogadó(k) számára.

- a *hívó művelet (call action)* egy olyan művelet, melyben egy üzenet példány generálódik. Ez az üzenet példány egy olyan operációt idéz elő mely meghívása a fogadón történik.

- a *helyi hívás (local invocation)* egy olyan művelet, mely egy operáció helyi végrehajtásához vezet.

- a *létrehozó művelet (create action)* egy olyan művelet, melyben egy példány jön létre. A létrehozás az osztályozók meghatározott halmazának definícióján alapul.

- a *megszüntető művelet (terminate action)* egy olyan művelet, melyben egy példány sajátmaga megszűnését idézi elő.

- a *leromboló művelet (destroy action)* egy olyan művelet, melyben egy példány egy másik példány megszűnését idézi elő.

- a *visszatérési művelet (return action)* egy olyan művelet, mely egy értéket ad vissza a hívónak.

- az *értelmezetlen művelet (uninterpreted action)* olyan művelet, melynek nincs interpretációja az UML -ben.

Minden műveletnek van egy specifikációja a cél objektum halmazban, mely nulla vagy több példányra bontja fel a példányokat amikor a művelet végrehajtódik. E példányok egy jel vagy egy operáció hívás címzettjei. Minden művelet rendelkezik még egy kifejezéslistával, mely felbontja azt az aktuális argumentum értékek listájára amikor a művelet végrehajtódik. Egy művelet mindig egy példány kontextusán belül hajtódik végre.

Egy művelet kérelmet küldhet el egy másik példányhoz (pl. *hívó művelet, küldő művelet*). A művelet meghatározza, hogy a fogadó és az argumentumok a kérelem minden elküldött példányához kiértékelődjenek. Továbbá a művelet még azt is meghatározza, hogy hány üzenet példányt kell elküldeni, valamint azt is megadja, hogy sorosan, vagy párhuzamosan (*ismétlődés*) kell őket elküldeni. Egy speciális esetben ez olyan feltétel specifikációjához is felhasználható, melynek teljesülnie kell abban az esetben, ha a kérelem elküldésre került; egyébként a kérelem nem kerül kiértékelésre.

.5 Szabványos elemek

Az előre definiált sztereotípusok, megszorítások és csatolt értékek az Általános Viselkedéshez a következő táblázatban látható, definícióját a *Szabványos Elemek* függelék tartalmazza.

Modell Elem	Stereotípus	Megszorítás	Csatolt Értékek
Példány			persistent
KapocsVégpont		association, global, local, parameter, self	
Kérelem		broadcast, vote	

Általános Viselkedés - Szabványos Elemek

2 Együttműködés

.1 Áttekintés

Az Együttműködés csomag a Viselkedési Elemek csomag egyik alcsomagja. Meghatározza azokat a fogalmakat, melyek szükségesek annak kifejezésére, hogy a különböző modellelemek miképpen állnak egymással kölcsönhatásban, mindezt struktúrális nézőpontból szemlélve. E jelen csomag az UML Alap-, illetve az Általános Viselkedés csomagjában definiált konstrukciót használja.

Az ***Együttműködés*** a ***ModellElem***ek egy meghatározott felhasználási módját definiálja egy ***Modell***ben. Leírja, hogy az ***Osztályozó***k és ***Asszociáció***ik különböző fajtái miképpen használhatók fel egy adott feladat megvalósításához. Az ***Együttműködés*** az ***Osztályozó***k egy ***Modell***jének korlátozását vagy vetületét definiálja, azaz a résztvevő ***Osztályozó***k milyen tulajdonságú ***Példány***ainak kell bennt lenniük egy adott ***Együttműködés***ben. Ugyanaz az ***Osztályozó*** vagy ***Asszociáció*** megjelenhet több ***Együttműködés***ben, illetve többször megjelenhet egy ***Együttműködés***ben, minden alkalommal más szerepben. Minden megjelenésben meghatározott, hogy az ***Osztályozó***, vagy ***Asszociáció*** mely tulajdonságai szükségesek egy adott felhasználáshoz. E tulajdonságok az ***Osztályozó*** vagy ***Asszociáció*** tulajdonságainak részhalmazát képezik. A ***Példány***ok és ***Kapcsok*** csoportja idomul azon szereplőkhöz, melyek az ***Együttműködés***ben kooperálnak egy meghatározott feladat végrehajtásakor. Ennélfogva az ***Osztályozó*** struktúra magában foglalja az illeszkedő ***Példány***ok lehetséges együttműködési struktúráit. Egy együttműködés szemléltethető diagrammal; vagy a résztvevő ***Osztályozó***k és ***Asszociáció***k korlátozott nézeteinek bemutatásával, vagy a korlátozott nézetekhez illeszkedő prototípus ***Példány***ok és ***Kapcsok*** ábrázolásával.

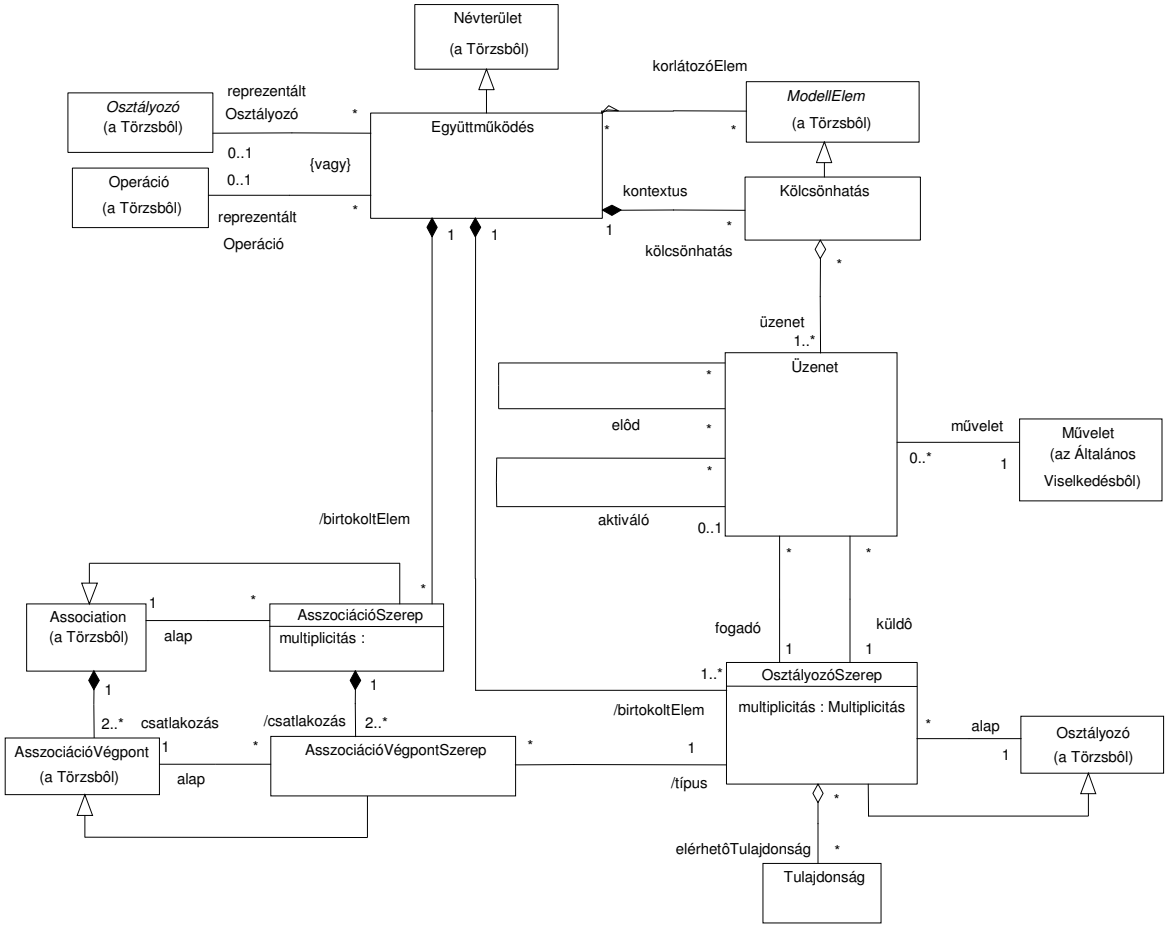
Az ***Együttműködés***ek több különböző dolog kifejezésére használhatók fel, mint pld. a használati esetek megvalósítása, a ROOM szereplő struktúrák, OORam szerep modellek és az együttműködések ahogy a Catalysis -ben definiálva vannak. Az ***Együttműködés***ek felhasználhatók még a ***Kölcsönhatás***ok kontextusának kialakításához, valamint egy ***Alrendszer*** specifikációs- és realizációs része közötti leképezés definiálásához.

Egy - az ***Együttműködés*** kontextusban definiált - ***Kölcsönhatás*** meghatározza azon kommunikációk részleteit, melyekre egy meghatározott feladat lebonyolításakor kerül sor. Leírja, hogy mely ***Kérelmek***et kell elküldeni, illetve e kérelmek belső sorrendjét is.

A következő szakaszok az Együttműködések csomag absztrakt szintaxisát, jól képzett szabályait és szemantikáját írják le.

.2 Absztrakt szintaxis

Az Együttműködések csomag absztrakt szintaxisa grafikus formában látható a 15. ábrán.



15. ábra: Együttműködések

Az Együttműködések csomag a következő metaosztályokat tartalmazza:

AsszociációVégpontSzerep (AssociationEndRole)

Az *asszociáció-végpont szerep* egy asszociáció végpontja, melyet egy együttműködésben használunk.

A metamodellben az ***AsszociációVégpontSzerep*** az ***AsszociációSzerep*** részét képezi és meghatározza az ***AsszociációSzerep OsztályozóSzerep***hez való csatlakozását. Összefüggésben áll az ***AsszociációVégpont***tal, deklarálva a megfelelő részt egy ***Asszociáció***n belül.

Attributumok (Attributes)

multiplicitás (multiplicity) A ***KapocsVégpont***ok száma, mely egy ***Együttműködésben*** szerepet játszik.

Asszociációk (Associations)

alap (base) Az ***AsszociációVégpontSzerep*** az ***AsszociációVégpont*** egy vetülete (projekciója).

AsszociációSzerep (AssociationRole)

Az *asszociáció szerep* egy asszociáció speciális felhaszálása, mely asszociáció az együttműködésben szükséges.

A metamodellben egy ***AsszociációSzerep*** egy az ***Együttműködés***ben használt ***Asszociáció*** szűkített nézetét specifikálja. Az ***AsszociációSzerep*** az ***AsszociációVégpontSzerep***ek halmazának egy kompozíciója; megfelel a bázis ***Asszociáció AsszociációVégpont***jainak.

Attributumok (Attributes)

multiplicitás (multiplicity) A ***Kapcsok*** száma, mely egy ***Együttműködésben*** szerepet játszik.

Asszociációk (Associations)

alap (base) Az ***AsszociációSzerep,*** mely az ***Asszociáció*** egy vetülete (projekciója).

OsztályozóSzerep (ClassifierRole)

Az *osztályozó serep* egy az együttműködésbeli résztvevő által betöltött speciális szerep. Az osztályozó egy szűkített nézetét specifikálja, az együttműködésbeli igény határozza meg.

A metamodellben az ***OsztályozóSzerep*** az ***Együttműködés*** egy résztvevőjét határozza meg, azaz egy szerep ***Példány*** idomul hozzá. Egy ***Tulajdonság***halmazt deklarál, mely részhalmazát képezi azon ***Tulajdonság***oknak, melyek az alap ***Osztályozó***ban rendelkezésre állnak. Az ***OsztályozóSzerep*** az ***AsszociációVégpontSzerep***en keresztül csatlakozhat az ***AsszociációSzerep***ek egy halmazához.

Attributumok (Attributes)

multiplicitás (multiplicity) A ***Példányok*** száma, mely egy ***Együttműködésben*** szerepet játszik.

Asszociációk (Associations)

elérhetőTulajdonság (availableFeature) Az ***Osztályozó Tulajdonságainak*** részhalmaza, mely az ***Együttműködésben*** használatos.

alap (base) Az ***OsztályozóSzerep,*** mely az ***Osztályozó*** egy vetülete (projekciója).

Együttműködés (Collaboration)

Az *együttműködés* leírja, hogy egy operáció vagy egy osztályozó, amilyen egy használati eset, hogyan jön létre az osztályozók és asszociációk halmaza által egy meghatározott módon. Az együttműködés kontextust definiál a kölcsönhatások által meghatározott feladatok végrehajtásához.

A metamodellben az ***Együttműködés*** az ***OsztályozóSzerep***ek és ***AsszociációSzerep***ek halmazát tartalmazza, mely azon ***Osztályozó***kat és ***Asszociáció***kat reprezentálja, melyek részt vesznek a kapcsolódó ***Osztályozó*** vagy ***Operáció*** realizálásában. Az ***Együttműködés*** tartalmazhatja azon ***Kölcsönhatás***ok halmazát is, melyek a ***Példány***ok által mutatott viselkedésnek a leírására használatosak. A ***Példány***ok viselkedése a résztvevő ***OsztályozóSzerep***ekhez igazodik.

Az ***Együttműködés*** az ***Osztályozó***k modelljének (szűkített-, hasított-, vetületi-) nézetét specifikálja. A vetület leírja a ***Példány***ok közötti szükséges kapcsolatokat, melyek idomulnak a résztvevő ***OsztályozóSzerep***ekhez; csakúgy, mint ezen ***Osztályozó***k ***Tulajdonság***ainak igényelt részhalmazát. Több ***Együttműködés*** az ***Osztályozó***k ugyanazon halmazának különböző vetületeit is leírhatja. Ennélfogva egy ***Osztályozó*** alapot képezhet több ***OsztályozóSzerep*** számára.

Az ***Együttműködés*** hivatkozhat a ***ModellElem***ek csoportjára is, rendszerint ***Osztályozó***kra és ***Általánosítás***okra, melyek a struktúrális követelmények kifejezéséhez szükségesek. Ilyenek az ***Általánosítás***ok, melyek maguk az ***Osztályozó***k között szükségesek, hogy megfeleljenek az ***Együttműködés*** szándékának.

Asszociációk (Associations)

korlátozóElem (constrainingElement) Azok a ***ModellElem***ek, melyek külön megszorításokat adnak. Ilyenek az ***Általánosítás*** és a ***Megszorítás*** azon ***ModellElem***eken, melyek az ***Együttműködés***ben résztvesznek.

kölcsönhatás (interaction) Azon ***Kölcsönhatások*** halmaza, melyek az ***Együttműködés***ben vannak definiálva.

birtokoltElem (ownedElem) (a *Névterületből* öröklődik) Az *Együttműködés* által meghatározott szerepek halmaza. Ezek *OsztályozóSzerepek* és *AsszociációSzerepek*.

reprezentáltOsztályozó (representedClassifier) Az az ***Osztályozó***, melynek az ***Együttműködés*** egy mevalósítása. (Akkor használatos, ha az ***Együttműködés*** egy ***Osztályozó***t reprezentál.)

reprezentáltOperáció (representedOperation) Az az ***Operáció***, melynek az ***Együttműködés*** egy mevalósítása. (Akkor használatos, ha az ***Együttműködés*** egy ***Operáció***t reprezentál.)

Kölcsönhatás (Interaction)

A *kölcsönhatás* azon üzeneteket specifikálja, melyeket a példányok küldenek egymásnak egy meghatározott feladat lebonyolítása érdekében.

A metamodellben a ***Kölcsönhatás Üzenet***ek halmazát tartalmazza, és ezáltal specifikálja a kommunikációt a ***Példány***ok között, igazodva a birtokolt ***Együttműködés OsztályozóSzerep***eihez.

Asszociációk (Associations)

kontextus (context) Az az ***Együttműködés***, mely a ***Kölcsönhatás*** kontextusát definiálja.

üzenet (message) Az az ***Üzenet***, mely specifikálja a kommunikációt a ***Kölcsönhatásban***.

Üzenet (Message)

Az *üzenet* defniálja egy meghatározott kérelem felhasználásának módját egy kölcsönhatásban.

A metamodellben az ***Üzenet*** egy ***Kölcsönhatás***beli ***Kérelem*** meghatározott felhasználását definiálja. Meghatározza a küldő és a fogadó szerepeit, csakúgy, mint a ***Művelet***ek elküldését. Továbbá meghatározza az ***Üzenet***ek relatív sorrendjét a ***Kölcsönhatás***on belül.

Asszociációk (Associations)

aktiváló (activator) Az az **Üzenet**, mely meghívta azt az operációt, melynek metódusa a kurrens **Üzenet**et tartalmazza.

alap (base) Az **Üzenet** specifikációja.

fogadó (receiver) A **Példány** azon szerepe, mely fogadja az *Üzenetet* és reagál rá.

előd (predecessor) Azon **Üzenet**ek halmaza, melyek befejezése lehetővé teszi a kurrens *Üzenet* végrehajtását. Az összes **Üzenet**nek be kell fejeződnie, mielőtt a végrehajtás elkezdődik. Üres abban az esetben, ha ez az első üzenet ametódusban.

küldő (sender) Annak a **Példány**nak a szerepe, mely az *Üzenetet* küldi és esetleg fogadja a választ.

.3 Jól képzett szabályok

A következő jól képzett szabályokat alkalmazzuk az Együttműködések csomaghoz:

AsszociációVégpontSzerep (AssociationEndRole)

[1] Az ***OsztályozóSzerep*** típusának igazodnia kell a bázis ***AsszociációVégpont*** típusához.

```
self.type = self.base.type
or
self.type.allSupertypes->includes (self.base.type)
```

[2] A típus az ***OsztályozóSzerep*** egy fajtája kell legyen.

```
self.type.oclIsKindOf (ClassifierRole)
```

AsszociációSzerep (AssociationRole)

[1] Az ***AsszociációVégpontSzerepeknek*** igazodniuk kell a bázis ***Asszociáció AsszociációVégpont***jaihoz.

```
Sequence{ 1..(self.role->size) }->forAll (index |
   self.role->at(index).base = self.base.connection->at(index))
```

[2] A végpontoknak az ***AsszociációVégpontSzerepek*** fajtájainak kell lenniük.

```
self.role->forAll( r | r.oclIsKindOf (AssociationEndRole) )
```

OsztályozóSzerep (ClassifierRole)

[1] Az ***OsztályozóSzerep***hez csatlakoztatott ***AsszociációSzerep***eknek illeszkedniük kell az ***Osztályozó***hoz kapcsolt ***Asszociáció***k részhalmazával.

```
self.allAssociations->forAll( ar |
   self.base.allAssociations->exists ( a | ar.base = a ) )
```

[2] Az ***OsztályozóSzerep Tulajdonság***ai a bázis ***Osztályozó Tulajdonság***ainak részhalmazát kell képezzék.

```
self.base.allFeatures->includesAll (self.availableFeature)
```

[3] Egy ***OsztályozóSzerep***nek nem lehetnek saját ***Tulajdonság***ai.

```
self.allFeatures->isEmpty
```

Együttműködés (Collaboration)

[1] Az ***Együttműködés***ben az ***OsztályozóSzerep***ek és ***AsszociációSzerep***ek összes ***Osztályozó***jának és ***Asszociáció***jának az ***Együttműködést*** bitokló névterületben kell lennie.

```
self.ownedElement->forAll ( e |
(e.oclIsKindOf (ClassifierRole) implies
   self.namespace.allContents->includes (e.oclAsType(ClassifierRole).base) )
and
(e.oclIsKindOf (AssociationRole) implies
   self.namespace.allContents->includes (e. oclAsType(AssociationRole).base) ))
```

[2] Az összes korlátozó ***ModellElem***nek az ***Együttműködés***t birtokló névterületben kell lennie.

```
self.constrainingElement->forAll ( ce |
   self.namespace.allContents->includes (ce) )
```

[3] Ha egy ***OsztályozóSzerep*** vagy egy ***AsszociációSzerep*** nem rendelkezik névvel, akkor csak egy lehet egy meghatározott bázissal.

```
self.ownedElement->forAll ( p |
   (p.oclIsKindOf (ClassifierRole) implies
      p.name = '' implies
         self.ownedElement->forAll ( q |
            q.oclIsKindOf(ClassifierRole) implies
               (p.oclAsType(ClassifierRole).base =
                  q.oclAsType(ClassifierRole).base implies p = q) ) )
and
   (p.oclIsKindOf (AssociationRole) implies
      p.name = '' implies
         self.ownedElement->forAll ( q |
            q.oclIsKindOf(AssociationRole) implies
               (p.oclAsType(AssociationRole).base =
                  q.oclAsType(AssociationRole).base implies p = q) ) )
)
```

[4] Egy ***Együttműködés*** csak ***OsztályozóSzerep***eket és ***AsszociációSzerep***eket tartalmazhat.

```
self.ownedElement->forAll ( p |
   p.oclIsKindOf (ClassifierRole) or
   p.oclIsKindOf (AssociationRole) )
```

Kölcsönhatás (Interacion)

[1] Az összes ***Jelnek***, mely az ***Üzenet***ek alapját képezi szerepelnie kell abban a névterületben, mely a ***Kölcsönhatás***t birtokolja.

```
self.message->forAll ( m |
   m.base.oclIsKindOf(Signal) implies
            self.collaboration.namespace.allContents->includes (m.base) )
```

Üzenet (Message)

[1] A küldőnek és a fogadónak részt kell vennie az ***Együttműködés***ben, mely a ***Kölcsönhatás*** kontextusát definiálja.

```
self.interaction.context.ownedElement->includes (self.sender)
and
self.interaction.context.ownedElement->includes (self.receiver)
```

[2] Az elődöknek és az aktiválóknak ugyanabban a ***Kölcsönhatás***ban kell szerepelniük.

```
self.predecessor->forAll ( p | p.interaction = self.interaction )
and
self.activator->forAll ( a | a.interaction = self.interaction )
```

[3] Az elődöknek ugyanazzal az aktiválóval kell rendelkezniük, mint az ***Üzenet***.

```
self.allPredecessors->forAll ( p | p.activator = self.activator )
```

[4] Egy ***Üzenet*** nem lehet saját elődje.

```
not self.allPredecessors->includes (self)
```

Kiegészítő operációk

[1] Az *összesElőd* operáció olyan halmazt ad eredményül, melynek elemei a kurrenset megelőző összes ***Üzenet***.

```
allPredecessors : Set(Message);
allPredecessors = self.predecessor->union (self.predecessor.allPredecessors)
```

.4 Szemantika

Ez a szakasz az Együttműködések csomag elemeinek szemantikáját írja le. Két részre bomlik: Együttműködés és Kölcsönhatás.

Együttműködés (Collaboration)

A következő szövegben az "*egy együttműködés példánya*" kifejezés azon példányok halmazát jelöli ki, melyek együttesen vesznek részt, illetve hajtanak végre egy meghatározott együttműködést.

Az *együttműködés* célja specifikálni, hogy egy operációt vagy egy osztályozót az osztályozók és asszociációk halmaza miképpen valósít meg. Az osztályozók és asszociációik együttesen vesznek részt az együttműködésben és igazodnak a megvalósított operáció vagy osztályozó követelményeihez. Az együttműködés egy olyan kontextust definiál, melyben a realizált elem viselkedése meghatározható az együttműködés résztvevői közötti kölcsönhatások kifejezései útján. Ily módon tehát, amíg egy modell egy egész rendszert ír le, az együttműködés annak a modellnek csupán egy szeletét, vagy vetületét adja. Definiálja tartalmának részhalmazát, mint osztályozókat és asszociációkat.

Egy együttműködés két különböző szinten prezentálható: specifikáció szinten vagy példány szinten. A specifikáció szintű diagram osztályozó- és asszociáció szerepeket; a példány szintű diagram példányokat és kapcsokat ábrázol, illeszkedve az együttműködés szabályaihoz.

Az együttműködésben meghatározott, hogy a tulajdonság példányoknak képeseknek kell lenniük részt venni az együttműködésben, azaz minden résztvevőnek, mely a szükséges tulajdonsághalmazt specifikálja, illeszkedő példánnyal kell rendelkeznie. Továbbá az együttműködés azt is megadja, mely asszociációknak kell jelen lenniük a résztvevők között. Nem mindig szükséges a résztvevő osztályozók összes tulajdonsága és ezen osztályozók közötti összes asszociáció egy adott együttműködésben. Ezért egy együttműködés valójában nem az osztályozó kifejezéseiben definiálva, hanem az *osztályozó szerepek*ében. Ily módon tehát, amíg egy osztályozó a példányok teljes leírása, egy osztályozó szerep azon tulajdonságok leírása, melyek egy meghatározott együttműködésben szükségesek; azaz egy osztályozó szerep egy osztályozó vetülete. A reprezentált osztályozó mint *bázis osztályozó* hivatkozik rá. Több osztályozó szerepnek lehet ugyanaz a bázis osztályozója, még ugyanabban az együttműködésben is, de tulajdonságaik az osztályozó tulajdonságainak különböző részhalmazait alkothatják. Ezek az osztályozó szerepek azután specifikálják azon különböző *szerepeket*, melyeket ugyanazon osztályozó példányai töltenek be.

Egy együttműködésben az *asszociáció szerepek* definiálják, hogy mely asszociációk szükségesek az osztályozók között e kontextusban. Minden asszociáció szerep egy együttműködésbeli asszociáció használatát reprezentálja és azon osztályozó szerepek között van definiálva, melyek a kapcsolódó osztályozókat reprezentálják. A feltüntetett asszociációt az asszociációs szerep *bázis asszociáció*jának nevezzük.

Egy példány, mely az együttműködés példányában vesz részt, meghatározott szerepet játszik, azaz idomul egy osztályozó szerephez az együttműködésben. A példányok számának szintén meghatározott szerepet kell játszania az együttműködés egy példányában. E számot (*multiplicitás*) az osztályozó szerep specifikálja. Különböző példányok játszhatják ugyanazt a szerepet, de az együttműködés különböző példányaiban. Mivel ezen példányok mindegyike ugyanazt a szerepet játssza, ezeknek illeszkedniük kell az osztályozó szerephez, mely a szerepet meghatározza. Ilymódon minden példánynak rendelkeznie kell attributum értékekkel, melyek megfelelnek az osztályozó szerep által specifikált attributumnak és szerepelniük kell azon kapcsokban, melyek az osztályozó szerephez csatlakoztatott osztályozó szerepeknek megfelelnek. Továbbá, egy osztály különböző szerepeket játszhat az együttműködés különböző példányaiban; valójában egy példány egy együttműködésnek ugyanabban a példányában több szerepet is betölthet.

Amennyiben az együttműködés egy operációt reprezentál, a kontextusnak magába kell foglalnia olyan dolgokat is, mint paraméterek, attributumok és osztályozókat, stb. (Az utóbbiakat az az osztályozó tartalmazza, melyet az operáció birtokol.) A kölcsönhatások specifikálják, hogy az argumentumok, az attributum értékek, a példányok, stb. hogyan fognak együttműködni az operáció által meghatározott viselkedés megvalósítása érdekében. Egy együttműködés felhasználható annak specifikálására, hogy egy operációt vagy egy osztályozót hogyan valósítson meg az együttműködő osztályozók csoportja. Egy olyan együttműködésben, mely egy operációt valósít meg, az alap osztályozók az operációk paraméter típusai az operációt birtokló osztályozó attributum típusaival együtt. Amikor az együttműködés egy osztályozót reprezentál, alap osztályozói lehetnek bármely fajtából valók, mint például az osztályok vagy az alrendszerek.

Az együttműködések specifikálják azt a kontextust, melyben a kölcsönhatások végrehajtódnak.

Egy együttműködés lehet egy *sablon* specifikációja. Az ilyen együttműködésből természetesen nem lesznek példányok, de könnyen felhasználható normál együttműködések generálására, melyek példányosíthatók. A sablon együttműködéseknek lehetnek paraméterei is, melyek helyfoglalókként funkcionálnak a sablonban. Általában ezeknek a paramétereknek osztályozóknak és asszociációknak kell lenniük, de a modell elemek más fajtái is definiálhatók mint együttműködésbeli paraméter. Ilyenek az operáció vagy a jel. Egy sablonból generált együttműködésben e paramétereket más modell elemek finomítják, melyek példányosíthatóvá teszik az együttműködést.

Továbbá egy együttműködés lehetnek *korlátozó modell elemei*, hasonló módon, mint ahogyan megszorításokkal és általánosításokkal, vagy esetleg más külön osztályozóval rendelkezik. E korlátozó modell elemek nem vesznek részt az együttműködésben. Arra szolgálnak, hogy külön megszorításokat fejezzenek ki a résztvevő elemeken abban az együttműködésben, melyre nem terjednek ki maguk a résztvevő szerepek. Például egy

sablonban szükséges lehet, hogy az osztályozók közül kettőnek közös őse legyen, vagy egy osztályozónak egy másik osztályozó alosztályának kell lennie. E követelményfajták nem fejezhetők ki asszociáció szerepekkel, mivel a szükséges kapcsokat fejezik ki a résztvevő példányok között. Ezért az együttműködéshez egy külön modell elem halmazt csatoltak.

Kölcsönhatás (Interaction)

A *kölcsönhatás* célja specifikálni a kommunikációt az egymással kölcsönhatásban álló példányok között egy meghatározott feladat végrehajtása érdekében. Egy kölcsönhatás az *együttműködésen* belül van definiálva, azaz az együttműködés definiálja azt a kontextust, melyben a kölcsönhatás elhelyezkedik. A példányok által lebonyolított kommunikációt a kölcsönhatás specifikálja, mely illeszkedik az együttműködés osztályozó szerepeihez.

Egy kölcsönhatás az *üzenet példányok* végrehajtását specifikálja. Ezek részben rendezettek, attól függően, hogy mely végrehajtási szálhoz tartoznak. A végrehajtást minden szál első üzenet példánya indítja el, miután azt elküldték. Minden végrehajtási szálon belül az üzenet példányok végrehajtása szekvenciális sorrendben történik, amíg a különböző szálak üzenet példányai párhuzamosan, vagy tetszőleges sorrendben is végrehajthatók.

A *kérelem* a példányok közötti kommunikáció specifikációja, mint amilyen egy hívó művelet, vagy egy küldő művelet. A kérelem megadja az operáció nevét, melyhez alkalmazták, vagy az esemény nevét, amely végrehajtásának igényét előidézte a fogadóban, csakúgy, mint az argumentumokban. Továbbá a kérelem meghatározza még a stimulus irányát is, azaz hogy a stimulus egy operáció meghívása, vagy egy válasz, valamint azt, hogy a stimulus szinkron, vagy aszinkron -e. Amennyiben aszinkron, akkor a példány az üzenet példány elküldése után közvetlenül folytatja végrehajtását, míg ha szinkron, akkor az üzenet példány elküldése után válaszra vár.

Az *üzenet* a kérelem használata egy kölcsönhatáson belül. Meghatározza a küldő és a fogadó típusát, csakúgy, mint azt, hogy mely üzeneteket kellett fogadni és melyeket elküldeni a kurrens üzenet előtt. Továbbá az üzenet még meghatározza a fogadó által várt választ is, melynek illeszkednie kell a fogadó megfelelő operációjának specifikációjával

A kölcsönhatás meghatározza minden üzenet *aktiválóját* és *elődjét*. Az *aktiváló* az az üzenet, mely meghívta az eljárást, melynek aktuális üzenete egy lépés. A kezdeti üzenet kivételével a kölcsönhatás minden üzenete rendelkezik egy aktiválóval. Az *elődök* egy olyan üzenethalmazt alkotnak, melyeknek be kell fejeződniük a kurrens üzenet végrehajtása előtt. Az eljárásban az első üzenetnek - értelemszerűen - nincsenek elődjei. Ha egy üzenetnek egynél több elődje van, akkor az a két vezérlési szál egybefolyását reprezentálja. Ha egy üzenetnek egynél több utóda (az előd inverze) van, akkor az a vezérlési szál szétágazását relzi (fork - join). Az *elődök* kapcsolat egy részbeni rendezettséget kényszerít rá az üzenetekre egy eljáráson belül, ezzel szemben az *aktiváló* kapcsolat egy fa szerkezetet kényszerít az operációk aktiválására. Az üzenetek végrehajthatók konkurens módon, de az *elődök* és *aktiválók* kapcsolat által kikényszerített szekvenciális megszorítások érvényesek rájuk.

Minden üzenet példány egy *művelet* végrehajtása által kerül elküldésre. A művelet meghatározza, hogy a fogadó és az argumentumok az üzenet minden elküldött példánya számára kiértékelődnek. Továbbá, a művelet azt is meghatározza, hogy iterációt vagy feltételt kell -e alkalmazni, illetve azt, hogy az iterációt szekvenciálisan, vagy párhuzamosan kell alkalmazni.

.5 *Szabványos elemek*

Jegyzet.

.6 *Megjegyzések*

A *Minta (Pattern)* egy szinoníma a sablon együttműködéshez, mely leírja egy tervezési minta struktúráját. A tervezési minták sok nem struktúrális aspektust foglalnak magukba, mint amilyen a heurisztika a használatukhoz, vagy az előnyök és hátrányok listái. Az ilyen aspektusokat nem modellezi az UML, ezért szövegesen, vagy táblázatokkal lehet őket reprezentálni.

3 Használati esetek

.1 Áttekintés

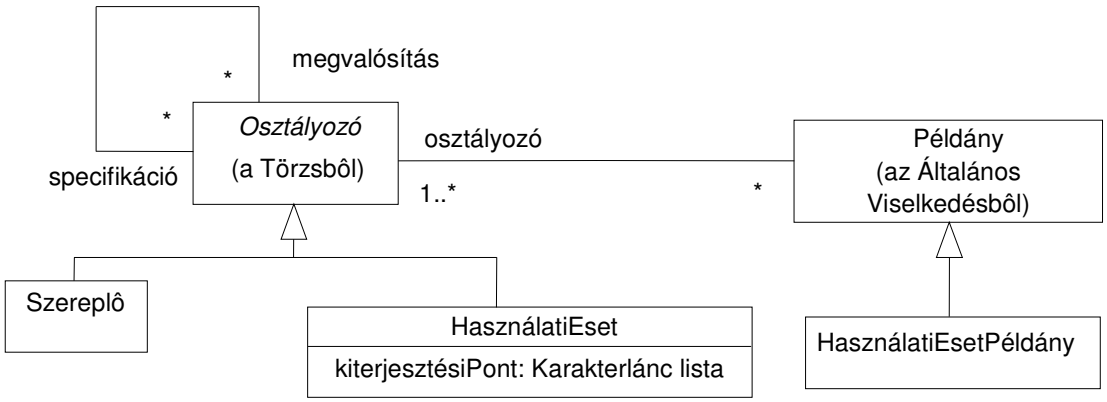
A Használati Esetek csomag a Viselkedési Elemek csomag egyik alcsomagja. Meghatározza azon fogalmakat, melyek egy egyed (rendszer) funkcionalitásának definíciójához használatosak. A csomag olyan konstrukciókat használ, melyek az UML Alap- illetve Általános Viselkedés csomagjában vannak definiálva.

A Használati Esetek csomag elemeit elsősorban egy egyed (rendszer, vagy alrendszer) viselkedésének definiálására használjuk a belső struktúra specifikálása nélkül. A kulcs elemek ebben a csomagban a ***HasználatiEset*** és a ***Szereplő***. A használati esetek és a szereplők pédányai kölcsönhatásban vannak, amikor az egyed szolgáltatásait igénybe veszik. Az egyeden belül levő osztályok definiálják a használati esetek megvalósításának módját az együttműködő objektumok kifejezésben, mindez megadható egy ***Együttműködés***sel. Egy egyed használati esete kifinomítható azon elemek használati eseteinek csoportjára, mely elemeket az egyed tartalmaz. Ezen alárendelt használati esetek kölcsönhatásának módja kifejezhető egy ***Együttműködés***ben is. Maga a rendszer funkcionalitásának specifikációját általában egy elkülönült használati eset modellben szokás kifejezni, azaz egy ***Modell***ben, mely sztereotípizált <<használati EsetModell>>.

A következő szakaszok a Használati Esetek csomag absztrakt szintaxisát, jól képzett szabályait és szemantikáját írják le.

.2 Absztrakt szintaxis

A Használati Esetek csomag absztrakt szintaxisának grafikus szemléltetése a 16. ábrán tekinthető meg.



16. ábra: Használati Esetek

A következő metaosztályokat tartalmazza a Használati Esetek csomag:

Szereplő (Actor)

A *szereplő* azon szerepek összefüggő halmazát definiálja, mely szerepeket egy egyed használói betölthetnek, amikor kölcsönhatásban állnak az egyeddel. Egy szereplő minden használati esethez rendelkezik egy - egy szereppel, mellyel kommunikál.

A metamodellben a ***Szereplő*** az ***Osztályozó*** alosztálya. Egy ***Szereplő*** *Névvel* rendelkezik és ***HasználatiEset***tekkel, valamint - megvalósítási szinten - ***Osztályozó***kkal kommunikálhat, miközben részt vesz ezen ***HasználatiEset***ek megvalósításában. Egy ***Szereplőnek Interfész***ei is lehetnek, ezek mindegyike leírja, hogyan kommunikálhatnak más elemek a ***Szereplő***vel.

Egy ***Szereplő*** örökíthet más ***Szereplő***ket. Ez azt jelenti, hogy az örökítő ***Szereplő*** képes lesz ugyanazokat a szerepeket játszani, mint az örökített ***Szereplő***, azaz kommunikálni ugyanazokkal a ***HasználatiEset***tekkel, mint az örökített ***Szereplő***.

HasználatiEset (UseCase)

A *haszálati eset* konstrukciót arra használjuk, hogy definiáljuk egy rendszer, vagy más szemantikus egyed viselkedését az egyed belső struktúrájának feltárása nélkül. Minden használati eset egy műveletsorozatot specifikál, beleértve a variánsokat is, melyeket az egyed képes végrehajtani kölcsönhatásban az egyed szereplőivel.

A metamodellben a ***HasználatiEset*** az ***Osztályozó*** alosztálya, ***Operáció***kat és ***Attributum***okat tartalmaz, specifikálja azon műveletsorozatokat, melyeket a ***HasználatiEset*** példánya hajt végre. A műveletek tartalmazzák az állapot megváltoztatását, és a ***HasználatiEset*** környezetével történő kommunikációt is.

A ***HasználatiEset***ek és a ***HasználatiEset***ek ***Szereplő***i között lehetnek ***Asszociáció***k. Egy ilyen ***Asszociáció*** megadja, hogy a ***HasználatiEset*** példányai és egy felhasználó kommunikálnak egymással. A ***HasználatiEset***ek összefüggésben állhatnak más ***HasználatiEset***tekkel is.

A ***HasználatiEset*** megvalósítását ***Együttműködések*** is specifikálhatják, azaz ***Együttműködés***ek definiálják, hogy a rendszerben a ***Példány***ok miként állnak egymással kölcsönhatásban a ***HasználatiEset*** sorozat végrehajtásához.

Attributumok (Attributes)

kiterjesztésiPont (extensionPoint) Egy karakterláncokból álló lista, mely a használati eseten belül definiált kiterjesztési pontokat reprezentálja. A kiterjesztési pont egy olyan hely, melynél a használati eset kiterjeszthető további viselkedéssel.

HasználatiEsetPéldány (UseCaseInstance)

A *használati eset példány* egy *használati esetben* specifikált műveletsorozat végrehajtása.

A metamodellben a ***HasználatiEsetPéldány*** a ***Példány*** alosztálya. Minden metódust egy ***HasználatiEsetPéldány*** hajt végre; ez utóbbi pedig elemi tranzakcióként hajtódik végre, azaz nem szakíthatja meg más ***HasználatiEsetPéldány***.

Egy explicit módon leírt ***HasználatiEsetPéldány***t *forgatókönyvnek (scenario)* hívunk.

.3 Jól képzett szabályok

A következő jól képzett szabályokat alkalmazzuk a Használati Esetek csomaghoz.

Szereplő (Actor)

[1] A ***Szereplő***knek csak ***Asszociáció***ik lehetnek a ***HasználatiEset***ekhez és az ***Osztály***okhoz, és ezek az ***Asszociáció***k binárisak.

```
self.associations->forAll(a |
   a.connection->size = 2 and
   a.allConnections->exists(r | r.type.oclIsKindOf(Actor)) and
   a.allConnections->exists(r |
      r.type.oclIsKindOf(UseCase) or
      r.type.oclIsKindOf(Class)))
```

[2] A ***Szereplő***k nem tartalmazhatnak ***Osztályozó***kat.

```
self.contents->isEmpty
```

[3] Egy felkínált ***Interfész***ben minden ***Operáció***hoz a ***Szereplő***nek rendelkeznie kell egy illeszkedő ***Operáció***val.

```
self.specification.allOperations->forAll (interOp |
self.allOperations->exists ( op | op.hasSameSignature (interOp) ) )
```

HasználatiEset (UseCase)

[1] A ***HasználatiEset***ek csak bináris ***Asszociáció***kkal rendelkezhetnek.

```
self.associations->forAll(a | a.connection->size = 2)
```

[2] A ***HasználatiEset***eknek nem lehetnek ***Asszociáció***ik olyan ***HasználatiEset***ekhez, melyek ugyanazon egyedet specifikálják.

```
self.associations->forAll(a |
   a.allConnections->forAll(s, o|
   s.type.specificationPath->isEmpty and o.type.specificationPath->isEmpty
   or
   (not s.type.specificationPath->includesAll(o.type.specificationPath) and
   not o.type.specificationPath->includesAll(s.type.specificationPath))
   ))
```

[3] Egy ***HasználatiEset***nek csak <<uses>> vagy <<extends>> ***Általánosítás***ai lehetnek.

```
self.generalization->forAll(g |
   g.stereotype.name = 'Uses' or g.stereotype.name = 'Extends')
```

[4] Egy ***HasználatiEset*** semmilyen ***Osztályozó***t nem tartalmazhat.

```
self.contents->isEmpty
```

[5] Egy felkínált ***Interfész***ben minden ***Operáció***hoz a ***HasználatiEset***nek rendelkeznie kell egy illeszkedő ***Operáció***val.

```
self.specification.allOperations->forAll (interOp |
   self.allOperations->exists ( op | op.hasSameSignature (interOp) ) )
```

Kiegészítő operációk

[1] A *specifikációÚtvonal (specificationPath)* operáció egy olyan halmazt ad eredményül, mely tartalmazza az összes környező ***Névterületet***, melyek nem példányai a ***Csomag***nak.

```
specificationPath : Set(Namespace)
specificationPath = self.allSurroundingNamespaces->select(n |
   n.oclIsKindOf(Subsystem) or n.oclIsKindOf(Class))
```

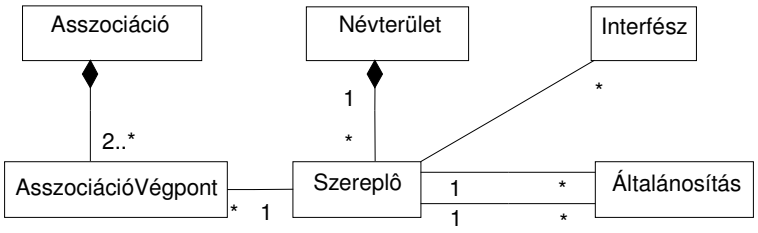
HasználatiEsetPéldány (UseCaseInstance)

Nincs külön jól képzett szabály.

.4 Szemantika

E szakasz leírja a Használati Esetek csoma elemeinek szemantikáját és más elemekkel való kapcsolataikat a Viselkedési Elemek csomagon belül.

Szereplő (Actor)



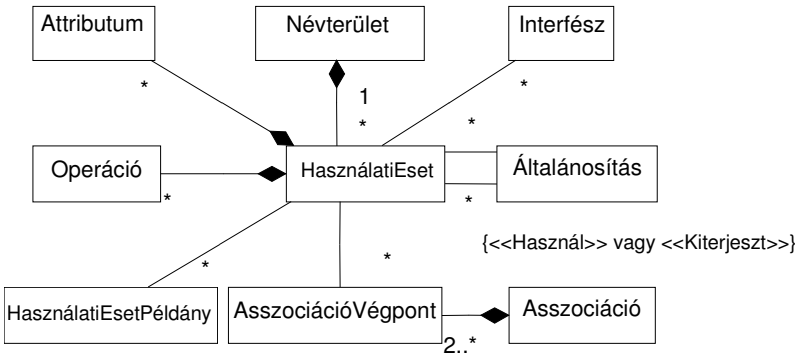
A *Szereplők* egy adott egyeden (mint például egy rendszer, alrendszer, vagy osztály) kívül elhelyezkedő modell részek, melyek kölcsönhatásban állnak az egyeddel. Minden szereplő egy összefüggő szerep halmazt definiál. E szerepeket az egyed használói tölthetik be, amikor kölcsönhatásban vannak az egyeddel. Minden alkalommal, amikor egy meghatározott felhasználó kölcsönhatásban áll az egyeddel, egy ilyen szerepet tölt be. A szereplő egy példánya egy speciális felhasználó, aki kölcsönhatásban áll az egyeddel. Egy olyan példány, mely idomul egy szereplőhöz, képes úgy működni, mint a szereplő egy példánya. Amennyiben az egyed egy rendszer, akkor a szereplők humán felhasználók és más rendszerek is lehetnek. Az alsóbb szintű alrendszer vagy osztály néhány szereplője egybeeshet a rendszer szereplőivel, amíg mások jelen vannak a rendszeren belül.

Mivel egy szereplő az egyeden kívül van, az ő belső struktúrája nem definiált, hanem csak a külső nézete, ahogyan az egyedből látszik. A szereplő példányok úgy kommunikálnak az egyeddel, hogy üzenet példányokat küldenek és fogadnak a használati eset példányoktól (példányoknak), valamint - a realizációs szinten - az objektumoktól (objektumoknak). Mindezt a szereplő és a használati eset vagy osztály közötti asszociációk fejezik ki.

Továbbá *interfészek* is csatlakozhatnak egy szereplőhöz, definiálva, hogy más elemek miképpen lehetnek kölcsönhatásban a szereplővel.

Két vagy több szereplő alkothat közösséget, azaz ugyanazokkal a használati esetekkel kommunikálhatnak azonos módon. Ennek a közösségnek a kifejezése más (esetleg absztrakt) szereplőre irányuló *általánosításokkal* történik, a szóbanforgó más szereplő modellezi ez esetben a közös szerep(ek)et. Egy örökös példánya mindig használható, ahol egy szülő példány szükséges.

HasználatiEset (UseCase)



A következő szövegben az *egyed* kifejezést akkor használjuk, amikor egy rendszerre, alrendszerre, vagy egy osztályra utalunk; a *modell elem* vagy *elem* pedig alrendszert, vagy osztályt jelent.

A *használati eset* célja definiálni az egyed egy viselkedését belső struktúrájának feltárása nélkül. Az eképpen meghatározott egyed lehet egy rendszer, vagy bármely modell elem, mely viselkedést tartalmaz, mint egy alrendszer, vagy egy osztály a rendszer modelljében. Minden használati eset egy szolgáltatást specifikál, melyet az egyed biztosít a felhasználók számára, azaz egy meghatározott felhaszálási módja az egyednek. Meghatároz egy felhaszáló által bevezetett teljes sorozatot, azaz a kölcsönhatásokat az egyed és a felhasználó között, csakúgy, mint az egyed által adott válaszokat, ahogyan azok a külvilágból érkeztek. Egy használati eset magába foglalja még e sorozat lehetséges variánsait is, pl. alternatív szekvenciák, kivételes viselkedés, hibakezelés, stb. A használati esetek teljes halmaza meghatározza az egyed összes különböző felhasználási módját, azaz az egyed összes viselkedése a használati eseteivel vannak kifejezve. E használati esetek praktikussági okokból *csomagok*ba szervezhetők.

Pragmatikus nézőpontból a használati esetek felhasználhatók egy egyed (külső) követelményeinek, és (a már megvalósított) egyed által felkínált funkcionalitás specifikálására egyaránt. Továbbá a használati esetek indirekt módon is megadják a specifikált egyed viselkedésére vonatkozó követelményeket, azaz hogyan létesítsenek kölcsönhatásokat, hogy szolgáltatásaikat képesek legyenek végrehajtani.

Mivel az egyedhez specifikált használati esetek felhaszálói mindig külsők, ezért reprezentálásuk az egyed szereplői által történik. Ilymódon tehát, ha a specifikált egyed egy rendszer vagy egy alrendszer a legfelső szinten, vagyis egy legfelső szintű csomag, akkor használati eseteinek felhasználóit a rendszer szereplői modellezik. Egy alsó szintű alrendszer vagy osztály azon szereplői, melyek a rendszer belsejét képezik, gyakran nincsenek explicit definiálva. Ehelyett a használati esetek közvetlenül a modell elemekhez tartoznak illeszkedve ezen implicit szereplőkhöz, vagyis melyek példányai betöltik e szerepeket a kölcsönhatásban a használati esetekkel. Ezen modell elemeket más csomagok vagy alrendszerek is tartalmazzák, ahol az alrendszer esetében tárolódhatnak a specifikációs részben, vagy a tartalomjegyzék részben is. A szereplő és az illeszkedő elemek közötti különbség gyakran elhanyagolható; így mindkettőre a *szereplő* kifejezéssel hivatkozunk.

Lehetnek olyan asszociációs kapcsolatok a használati esetek és a szereplők között, melyek azt jelentik, hogy a használati eset és a szereplő példányai kommunikálnak egymással. Egy szereplő egy egyed több használati esetével kommunikálhat, azaz a szereplő az egyed több szolgáltatását is kérheti, és egy használati eset egy vagy több szereplővel kommunikálhat amikor ezen szereplők szolgáltatásaikat nyújtják. Megjegyzendő, hogy két, ugyanazt az egyedet specifikáló haszálati eset nem kommunikálhat egymással, mivel mindegyikük önállóan írja le az egyed egy teljes használatát. Továbbá, a használati esetek minden alkalommal jeleket használnak fel, amikor a rendszeren kívüli szereplőkkel kommunikálnak, míg a rendszer belső elemeivel történő kommunikációik során más kommunikációs szemantikát is felhasználhatnak.

A szereplők és használati esetek közötti kölcsönhatás *interfészekkel* definiálható. Az interfész akkor az egész - használati esetben definiált - kölcsönhatás egy részhalmazát

definiálja. Ugyanazon használati eset különböző interfészeket kínálhat fel, melyeket nem szükséges feltétlenül szétválasztani.

A *használati eset példány* a használati eset egy példánya, melyet egy szereplő példányától származó üzenet vezet be. Az üzenetre adott válaszként a használati eset példány egy műveletsorozatot hajt végre, ahogyan azt a használati eset meghatározta, úgy, hogy üzeneteket küld el a szereplő példányokhoz. A szereplő példányok küldhetnek új példányokat a haszálati eset példánynak és a kölcsönhatás addig folytatódik, amíg a példány nem válaszolt az összes bemenetre, és nem vár több inputra, amikor befejezi ténykedését. Minden metódust egy használati eset példány hajt végre, mint elemi tranzakciót, melyet nem szakít meg más használati eset példány.

Egy használati eset leírható szövegesen, operációk használatával, művelet (aktivitás) diagrammal, állapotgép által, vagy egyéb viselkedés leíró technikákkal, amilyenek például az elő- és utófeltételek. A használati eset és a szereplők közötti kölcsönhatás még együttműködési diagramban is szemléltethető.

Abban az esetben, ahol az alrendszereket a csomaghierarchia modellezéséhez használjuk, a rendszer minden szinten specifikálható használati esetekkel, mivel a használati esetek minden alrendszer és minden osztály specifikációjához felhasználhatók. Egy modell elemet specifikáló használati eset gyakran finomítható kisebb használati esetek csoportjára, melyek mindegyike egy - az elsőben lévő - modell elem egyik szolgáltatását specifikálja. A fölérendelt használati eset által specifikált funkcionalitás teljes mértékben átmásolható az alárendelt használati eseteihez. Megjegyzendő persze, hogy a konténer elem struktúráját nem fedik fel a használati esetek, mivel azok csak az elem által felkínált funkcionalitást határozzák meg. Egy meghatározott fölérendelt használati eset minden alárendelt haszálati esettel együttműködik a fölérendelt végrehajtásához. Az ő kooperációjukat az *együttműködések* specifikálják, és együttműködési diagramban szemléltethető. Egy fölérendelt használati eset összes szereplője az alárendelt használati esetek szereplőikként jelenik meg. A kooperáló alárendelt használati esetek egymás szereplői. Továbbá egy fölérendelt használati eset interfészei lemásolhatók azon alárendelt használati esetek interfészeihez, melyek kommunikálnak a szereplőkkel. E szereplők a fölérendelt használati eset szereplői is egyben.

Az alárendelt használati esetek környezete az a modell elem, mely tartalmazza ezen használati esetek által specifikált modell elemeket. Így tehát alulról felfelé szemlélve az alárendelt használati esetek kölcsönhatása egy fölérendelt használati esetet eredményez, azaz a konténer elem egy használati esetét.

Az osztályok használati esetei az osztályok operációinak kifejezéseiben vannak specifikálva, mivel egy osztály egy szolgáltatása lényegében az osztály szolgáltatásainak meghívása. Néhány használati eset állhat egyetlen operáció alkalmazásából, míg mások egy egész operáció-halmazt is magukba foglalhatnak, esetleg egy jól definiált sorrendben. Egy operációt az osztály több szolgáltatása is igényelhet, és éppenezért az az osztály több használati esetében is megjelenhet.

Egy használati eset megvalósítása függ az őt specifikáló modell elem fajtájától. Például, mivel egy osztály használati eseteit az operációk eszközei specifikálják, megvalósításuk a megfelelő metódusok által történik, amíg egy alrendszer használati eseteit az alrendszerben levő elemek valósítják meg. Mivel egy alrendszernek nem lehet saját

viselkedése, minden - az alrendszer által felkínált - szolgáltatásnak az alrendszerben lévő elemek (végül is osztályok) által felkínált szolgáltatás kompozíciójának kell lennie. Ezek az elemek fognak együttműködni és együttesen végrehajtani a specifikált használati eset viselkedését. Egy vagy több együttműködés leírja egy használati eset megvalósítását. Ennél fogva az együttműködések egy használati eset finomítására és megvalósítására egyaránt felhasználhatók.

A használati esetek alkalmazása minden szinten nem csak a funkcionalitás (összes szintbeli) specifikációjának egységes módját foglalja magába, hanem egyben egy hatékony módszer a rendszer követelményeinek nyomon követésére is. Egy egyedi operáció módosítása hatásának továbbterjedése - a rendszer csomag viselkedését - tekintve hasonló módon követhető nyomon.

.5 Szabványos elemek

Ld. a *Szabványos Elemek* függeléket a <<kiterjeszt>> és a <<használatiEsetModell>> sztereotípusokhoz.

.6 Megjegyzés

Az alkalmazás egy pragmatikus szabálya a használati esetek definiálása során az, hogy minden használati esetnek néhány fajta eredmény értéket kell szolgáltatnia a szereplői (de legalább egy szereplője) számára. Ez biztosítja azt, hogy a használati esetek teljes specifikációk és nem csak töredékek.

4 Állapot Gépek

.1 Áttekintés

Az Állapot Gépek csomag a Viselkedési Elemek csomag alcsomagja. Meghatározza azon fogalmakat, melyek a véges állapot-átmenten keresztül a viselkedés modellezéséhez használatosak. Az Alap csomag kidolgozásaként van definiálva. Az Állapot Gépek csomag függ még azon fogalmaktól is, melyek az Általános Viselkedés csomagban kerültek definiálásra, lehetővé téve ezzel más Viselkedési Elemek -beli alcsomaggal való integrációt.

A metamodell az állapot ábrák objektum változatát támogatja. Az állapot ábrák számos fogalmi rövidítéssel vannak jellemezve, mint amilyenek a hierarchikus állapotok, konkurens állapotok, történet és elágazási csomópontok, melyek - kombinálva - a specifikációk jelentős tömörítését teszik lehetővé a legtöbb más állapot alapú

formalizmus felett. Értelemszerűen minden más véges állapot gép modell az állapotábrák korlátozott verzióinak tekinthető.

Az állapot gépek két különböző módon alkalmazhatók. Az egyik esetben az állapot gép környezetének, tipikusan egy osztálynak a teljes viselkedését specifikálhatja. Abban az esetben, amikor a kérelmezők elküldik kérelmeiket egy állapot gép tulajdonosához és az állapot gép fogad egy eseményt, meghatározza, hogy a hatás a műveleteknek azon átmenetekhez való hozzáillesztése által jön létre, melyekből az operációk teljes specifikációi származhatnak.

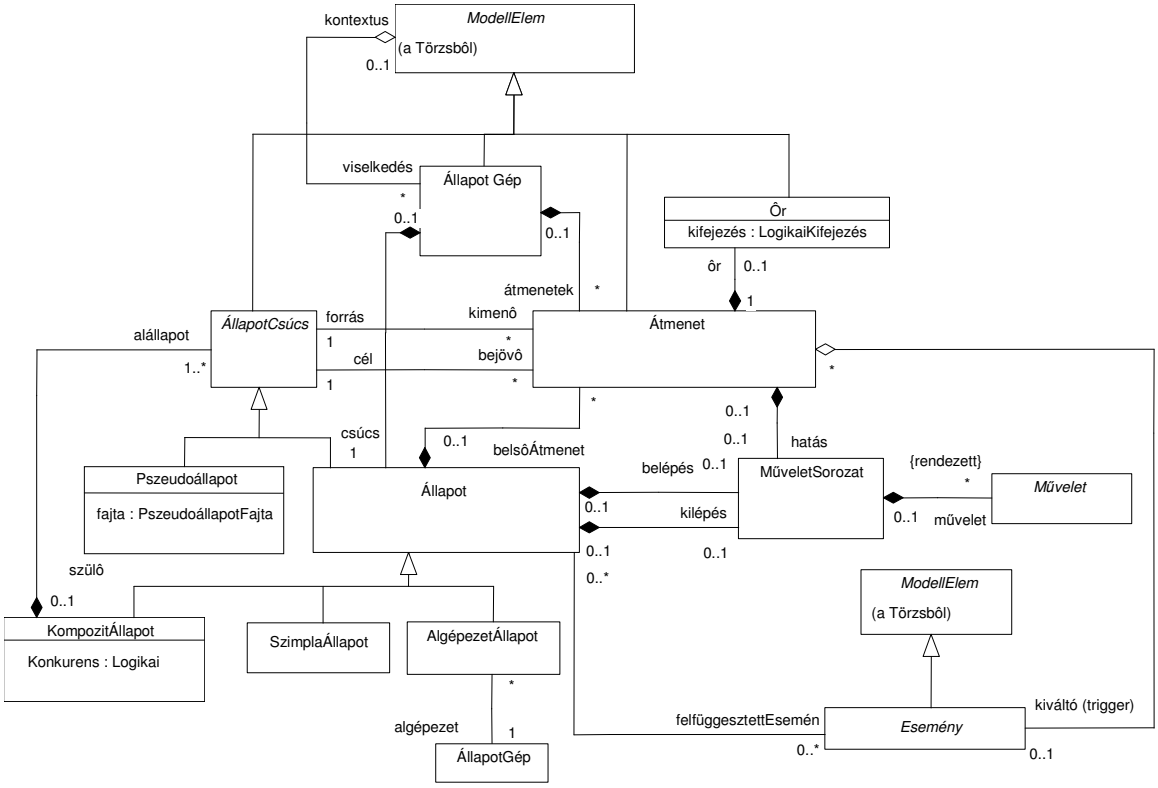
A második esetben az állapot gépek protokoll specifikációra alkalmazhatók, bemutatva azt a sorrendet, melyben az operációk meghívhatók. Az átmenetek azáltal aktivizálódnak, hogy meghívjuk az eseményeket, és azok műveletei meghívják a kívánt operációt. Ez azt jelenti, hogy egy hívó lehetővé teszi az operáció meghívását az adott pontnál. A protokoll állapot gép nem specifikál olyan műveleteket melyek magának az operációnak a viselkedését határozzák meg, de bemutatja egy állapot megváltozását, meghatározva a következő meghívható operációkat.

Továbbá az Állapot Gépek definiálásához a metamodell az aktivitás modellek alap szemantikáját is definiálja. Az állapotábrák és az aktivitás modellek több elemet is megosztanak, és ennélfogva ugyanarra a metamodellre, mint alapra épülnek. Az aktivitás modellek az állapot modellek altípusai, melyek a legtöbb olyan fogalmat használják, melyeket az állapot gépekhez alkalmazunk.

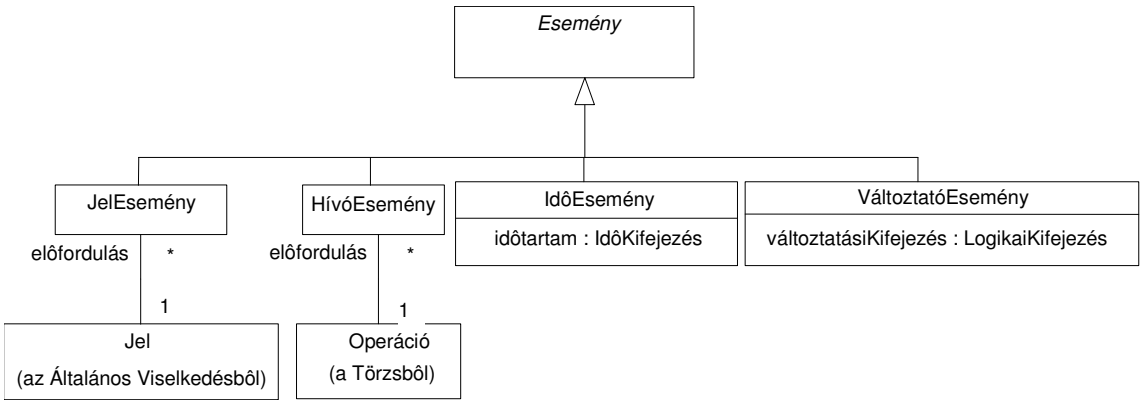
A következő szakaszok leírják az Állapot Gépek csomag absztrakt szintaxisát, jól képzett szabályait és szemantikáját.

.2 *Absztrakt szintaxis*

Az Állapot Gépek csomag absztrakt szintaxisának grafikus ábrázolása a 17. és 18. ábrán látható.



17. ábra: Állapot Gépek - Fő ábra



18. ábra: Állapot Gépek - Események

Az Állapot Gépek csomag a következő metaosztályokat tartalmazza:

HívóEsemény (CallEvent)

A *hívó esemény* nem más, mint egy kérelem fogadása egy operáció meghívásához. A várt eredmény az operáció végrehajtása.

A metamodellben a ***HívóEsemény*** az ***Esemény*** alosztálya, mely az az absztrakt metaosztály, mely reprezentálja az összes állapotgépbeli átmenetet kiváltó esemény típust.

A ***HívóEsemény*** két speciális esete az objektum létrehozó és az objektum leromboló esemény.

Asszociációk (Associations)

operáció (operation) Kijelöli azt az operációt, melynek a meghívása szükséges.

VáltoztatóEsemény (ChangeEvent)

Egy *változtató esemény* olyan esemény, mely akkor generálódik, amikor egy vagy több attributum vagy kapcsolat megváltoztatja értékét egy explicit kifejezésnek megfelelően.

Változtató esemény sohasem jön létre explicit változtató exemény művelet által. Ehelyett ez egy vagy több művelet végrehajtásának következménye, mely műveletek azon elemek értékeit módosítják, melyekre logikai kifejezés hivatkozik. A megfelelő változtató eseményt valójában az alárendelt futásidejű rendszer idézi elő, mely érzékeli, hogy a feltétel igazra változott.

Egy változtató esemény az átmenetekhez kiváltóként (triggerként) funkcionál, és nem szabad összetéveszteni egy őrrel (ez utóbbit ld. később). Amikor egy változtató esemény fordul elő, akkor egy őr képes blokkolt állapotban tartani bármely olyan átmenetet, melyet egyébként elindított (triggerelt) volna egy változtatás.

A metamodellben a ***VáltoztatóEsemény*** az ***Esemény*** alosztálya, mely absztrakt osztály, és reprezentálja az összes olyan eseményt, ami triggerel egy ***ÁllapotGép***et.

Attributumok (Attributes)

változtatásiKifejezés (changeExpressiono) Egy logikai kifejezés, mely jelez egy ***VáltoztatóEsemény*** előfordulásakor.

KompozitÁllapot (CompositState)

Egy *kompozit állapot* olyan állapot, mely alállapotokból áll.

A metamodellben egy ***KompozitÁllapot*** az ***Állapot*** alosztálya, egy vagy több alállapotot tartalmaz, mely(ek) az ***ÁllapotCsúcs*** altípusai.

Asszociációk (Associations)

alállapotok (substates) Kijelöli azon ***Állapotok*** halmazát, melyek egy ***KompozitÁllapot*** alállapotait alkotják. Minden alállapotot egyedileg birtokolja a szülő ***KompozitÁllapot***a.

Attributumok (Attributes)

konkurens (isConcurrent) Egy logikai érték, mely meghatározza a dekompozíció szemantikát. Ha ez az attributum igaz, akkor a kompozit állapot

közvetlenül két vagy több ortogonális összekötő komponensre bomlik szét (általában konkurens végrehajtással kapcsolódva). Ha ez az attributum hamis, akkor nincs közvetlen ortogonális komponens a kompozitban.[9] Ez azt jelenti, hogy pontosan egy alállapot lehet aktív egy adott pillanatban (szekvenciális végrehajtás).

régió (isRegion) Egy származtatott logikai érték, mely jelzi, hogy egy ***KompozitÁllapot*** a konkurens állapot alállapota. Ha kiértékelése igaz értéket ad eredményül, akkor a ***KompozitÁllapot*** a konkurens állapot alállapota.

Esemény (Event)

Egy *esemény* egy szignifikáns előfordulás specifikációja, melynek időbeli elhelyezkedése és térbeli helye van. Egy esemény példánya egy objektumon belüli viselkedési tulajdonság aktiválásához vezethet.

Fontos különbséget tenni egy esemény, mely egy statikus specifikáció egy dinamikusan előforduló fogalomhoz; és egy esemény aktuális példánya, mint egy program végrehajtás eredménye között. Az osztály ***Esemény*** reprezentálja egy esemény típusát. Egy esemény példánya nincs modellezve explicit módon a metamodellben.

A metamodellben egy ***Esemény*** a ***ModellElem*** alosztálya és része egy ***Átmenet***nek, mely az ő *trigger*ét reprezentálja.

Őr (Guard)

Egy *őr (guard)* feltétel egy logikai kifejezés, mely rendre hozzákapcsolható egy átmenethez annak meghatározására, hogy az átmenet engedélyezett legyen, vagy sem.

Az őr akkor értékelődik ki, amikor egy esemény előfordulás triggereli az átmenetet. Az esemény csakis akkor van jelen, ha az őr igaz az adott időpillanatban, és az állapot gép átmenet valójában ekkor fog érvényre jutni. Az őröknek tiszta kifejezéseknek kell lenniük, mellékhatások nélkül. A mellékhatásokkal járó őr kifejezések kiszámíthatatlan eredményekhez vezethetnek.

A metamodellben az ***Őr*** egy ***ModellElem***, így helyettesíthető a finomított állapot gépekben.

Attributumok (Attributes)

kifejezés (expression) Egy logikai kifejezés, mely meghatározza az őr feltételt.

PszeudoÁllapot (PseudoState)

Egy *pszeudoállapot* a csomópontok különböző típusainak absztrakciója az állapot gép grafikonon belül, mely átmenet pontokat reprezentál az átmeneti útvonalban egyik állapottól a másikhoz (azaz ág és elágazási pontok). A pszeudoállapotokat arra használjuk, hogy szimpla átmenetekből összetett átmeneteket képezzünk.

[9] Azonban lehet ortogonális komponens néhány tartalmazott állapotban.

A metamodellben a ***PszeudoÁllapot*** az ***ÁllapotCsúcs*** alosztálya, mely minden állapotábra csomópontot általánosít.

Attributumok (Attributes)

fajta (kind) Meghatározza a ***PszeudoÁllapot*** típusát és a következők egyike lehet: initial (kezdeti), deepHistory (részletesTörténet), shallowHistory (vázlatosTörténet), join (egyesülés), fork (elágazás), branch (ág), final (végső).

JelEsemény (SignalEvent)

Egy *JelEsemény* olyan eseményeket reprezentál, melyeket egy jel objektum általi fogadása eredményez.

A metamodellben a ***JelEsemény*** az ***Esemény*** alosztálya.

Asszociációk (Associations)

jel (signal) Kijelöli azt a ***Jelet***, melynek az állapot tulajdonosa általi fogadása kiválthat (triggerelhet) egy ***Átmenetet***.

SzimplaÁllapot (SimpleState)

A *SzimplaÁllapot* olyan állapot, melynek nincsenek alállapotai.

A metamodellben egy ***SzimplaÁllapot*** az ***Állapot*** alosztálya, melynek semmilyen további tulajdonsága nem lehet. Kizárólag a ***KompozitÁllapot***tal való szimmetriához alkalmazzuk.

Állapot (State)

Az *Állapot* egy feltétel vagy szituáció egy objektum élettartama folyamán, mialatt kielégít néhány feltételt, végrehajt néhány műveletet, vagy várakozik néhány esemény bekövetkezésére. Egy állapot egy dinamikus szituációt modellez, melyben jellegzegesen egy vagy több (implicit vagy explicit) feltételt tart fenn.

A metamodellben egy ***Állapot*** az ***ÁllapotCsúcs*** alosztálya, mely által öröklődnek a bejövő és kimenő ***átmenetek*** alapvető tulajdonságai, kapcsolódva az állapot csúcsokkal.

Asszociációk (Associations)

felfüggesztettEsemény (deferredEvent) ***Események*** listája, melyek előfordulásai az adott ***Állapot*** időtartama alatt elhalasztódnak, egészen addig, amíg a tulajdonos be nem lép egy olyan ***Állapotba***, melyben nincsenek felfüggesztve, ebben az időpontban már triggerelhetnek ***Átmeneteket***, mintha éppen akkor fordulnának elő.

belépés (entry) Egy opcionális ***MűveletSorozat***, ami egy ***Állapot***ba való belépés során hajtódik végre. Ezek a ***Műveletek*** elemiek, nem kerülhetők ki, és megelőznek minden belső műveletet vagy ***Átmenetet***.

kilépés (exit) Egy opcionális ***MűveletSorozat***, ami egy ***Állapot***ból való kilépés során hajtódik végre. Ezek a ***Műveletek*** elemiek, nem kerülhetők ki, és megelőznek minden belső műveletet vagy ***Átmenetet***.

belsőÁtmenet (internalTransition) ***Átmenetek*** halmaza mely átmenetek teljes mértékben az ***Állapot***on belül fordulnak elő. Ha triggereik közül egy kielégített, akkor a művelet az ***Állapot*** megváltoztatása nélkül végrehajtódik. Ez azt jelenti, hogy az ***Állapot***belépési vagy kilépési feltétele nem kerül meghívásra. Ezeket az ***Átmeneteket*** még akkor is alkalmazzuk, ha az ***ÁllapotGép*** egy beágyazott régióban van és ugyanabban az ***Állapot***ban marad.

felfüggesztettEsemény (deferredEvent) Egy asszociáció, mely meghatározza, hogy az ***Események*** felfüggesztettek legyenek, amennyiben az ***Állapot***on belül fogadják őket. A '*..*' multiplicitás jelzi, hogy egy ***Állapot*** több ***Eseményt*** képes felfüggeszteni, valamint egy ***Esemény***t felfüggeszthet több ***Állapot*** is.

ÁllapotGép (StateMachine)

Az *állapot gép* egy viselkedés, mely egy objektum vagy kölcsönhatás állapotainak sorozatát specifikálja. Ezen állapotokon megy keresztül élettartama során, válaszaival és műveleteivel együtt reagál az eseményekre. A viselkedést az állapot csomópontok gráfjának keresztezéseként specifikálták. Ezeket az állapot csomópontokat egy vagy több egyesített átmenet ív köti össze. Az átmeneteket esemény példányok sorozata váltja ki (triggereli).

A metamodellben egy ***ÁllapotGép*** ***Állapot***okból és ***Átmenet***ekből áll össze. A ***ModellElem*** szerep biztosítja a *kontextust* az ***ÁllapotGép***hez. A kontextus viszony egy általános esete az, ahol egy ***ÁllapotGép***et az ***Osztályozó*** *életciklusának* specifikálásához jelölünk ki. Az ***ÁllapotGép***nek vagy egy kompozíció aggregációja egy ***Állapot***hoz, mely reprezentálja a *csúcs* állapotot és az ***Átmenet***ek csoportját. Mint következmény, az ***ÁllapotGép*** birtokolja ***Átmenet***eit és (gráfbeli) csúcs[10] ***Állapot***át, de a beágyazott állapotokat tranzitíven birtokolja a szülő ***Állapot***ain keresztül.

Asszociációk (Associations)

kontextus (context) Kapcsolat egy ***ModellElem***hez. A birtokló ***ModellElem*** az az elem, melynek viselkedését az ***ÁllapotGép*** specifikálta. A ***ModellElem*** több ***ÁllapotGép***et is tartalmazhat. Minden ***ÁllapotGép***et egy ***ModellElem*** birtokol.

legfelső (top) Kijelöli azt a legmagasabb szintű ***Állapotot***, melyet közvetlenül birtokol az ***ÁllapotGép***. A többi ***Állapotot*** a szülő kompozit állapotok birtokolják. Az 1 multiplicitásnál egy ***Állapot***nak kell kijelöltnek lennie, mint legfelső ***Állapot***. Az ***ÁllapotGép*** fennmaradó része a ***KompozitÁllapot*** egy kiterjesztése.

átmenetek (transitions) Összekapcsolja az ***ÁllapotGép***et az ***Átmeneteivel***. Megjegyzendő, hogy a belső ***Átmeneteket*** az ***Állapot*** birtokolja, és

[10] Az eredeti angol szakirodalom a gráfbeli csúcsot "Vertex" -ként említi.

nem az ***ÁllapotGép***. Mnden más ***Átmenetet***, - melyek lényegében kapcsolatok az ***Állapotok*** között - az ***ÁllapotGép*** birtokol. A multiplicitás '0..*'.

ÁllapotCsúcs (StateVertex)

Az *ÁllapotCsúcs* egy csomópont absztrakciója egy állapot ábrában. Általában bármennyi átmenet forrása vagy célja lehet.

A metamodellben egy ***ÁllapotCsúcs*** a ***ModellElem*** egy alosztálya.

Asszociációk (Associations)

kimenő (outgoing) Meghatározza az átmeneteket, melyek a csúcsból indulnak ki.

bejövő (incoming) Meghatározza az átmeneteket, melyek a csúcsba érkeznek.

AlgépÁllapot (SubmachineState)

Az *AlgépÁllapot* egy beágyazott állapot gépet reprezentál. Egy beágyazott állapot gép szemantikusan egyenértékű egy kompozit állapottal, de elősegíti az újra felhasználást és a modularitást egy független beágyazott állapot gép formájában.

A metamodellben egy ***AlgépÁllapot*** az ***Állapot*** alosztálya.

Asszociációk (Associations)

algép (submachine) Az alállapot gépet reprezentálja.

IdőEsemény (TimeEvent)

Az *IdőEsemény* az ***Esemény*** altípusa. Azon esemény pédányok modellezésére szolgál, mely példányokat egy határidő lejárta eredményez.

A metamodellben egy idő esemény specifikálhatja egy átmenet triggerét, mely alapértelmezésként azt az időt jelenti, mely a kurrens állapotba való belépés óta eltelt.

Attributumok (Attributes)

időtartam (duration) Meghatározza a megfelelő határidőt.

Átmenet (Transition)

Az *Átmenet* egy kapcsolat egy *forrás* állapot csúcs és egy *cél* állapot csúcs között. Részét képezheti egy *összetett átmenetnek* , mely áthelyezi az állapot gépet az egyik állapot konfigurációból a másikba, reprezentálva az állapot gép teljes válaszát egy meghatározott esemény példányra egy adott forrás állapot konfigurációhoz.

A metamodellben az ***Átmenet*** azon Modell Elem alosztálya, mely különböző kapcsolatokban vesz részt más állapot gép metaosztályokkal:

Asszociációk (Associations)

kiváltó (trigger) Egy egyszerű ***Eseményt*** specifikál, mely aktiválta.

őr (guard) Kimondja, hogy igaz értékűre kell kiértékelni abban a pillanatban, amikor az ***Átmenet*** aktivizálódik (a trigger által).

hatás (effect) Meghatároz egy ***MűveletSorozatot***, melyet végre kell hajtani az ***Átmenet*** aktivizálásakor.

forrás (source) Kijelöli az ***ÁllapotCsúcs***ot, melyre az ***Átmenet*** aktiválása hat. Ha az ***ÁllapotCsúcs*** a forrás ***Állapot***ban van és az ***Átmenet*** triggerének feltétele teljesül, akkor aktivizálja, végrehajtja a ***Műveleteit***, és az ***ÁllapotGép*** a cél ***Állapot***ba lép.

cél (target) Kijelöli az ***ÁllapotCsúcs***ot, melyet az ***Átmenet*** aktiválása eredményezett, amikor az ***ÁllapotGép*** eredetileg a forrás ***Állapot***ban volt. Aktivizálás után az ***ÁllapotGép*** a cél ***Állapot***ban van.

.3 *Jól képzett szabályok*

A következő jól képzett szabályokat alkalmazzuk az Állapot Gépek csomaghoz.

KompozitÁllapot (CompositeState)

[1] Egy kompozit állapotnak legfeljebb egy kezdeti csúcsa lehet

```
self.subState->select (v | v.oclType = Pseudostate)->
   select(p : Pseudostate | p.kind = #initial)->size <= 1
```

[2] Egy kompozit állapotnak legfeljebb egy részletes történet csúcsa lehet

```
self.subState->select (v | v.oclType = Pseudostate)->
   select(p : Pseudostate | p.kind = #deepHistory)->size <= 1
```

[3] Egy kompozit állapotnak legfeljebb egy vázlatos történet csúcsa lehet

```
self.subState->select(v | v.oclType = Pseudostate)->
   select(p : Pseudostate | p.kind = #shallowHistory)->size <= 1
```

[4] Egy *konkurens* kompozit állapotban legalább két kompozit alállapotnak kell lenni

```
(self.isConcurrent) implies
   (self.subState->select (v | v.oclIsKindOf(CompositeState))->size <= 2)
```

Őr (Guard)

[1] Egy őrnek nem lehetnek mellékhatásai.

HelyiHívás (LocalInvocation)

[1] Egy helyi hívásnak nem lehet célja.

```
self.target->size = 0
```

PszeudoÁllapot (PseudoState)

[1] Egy kezdeti csúcsnak legfeljebb egy kimenő átmenete lehet, és nem lehet bejövő átmenete

```
(self.kind = #initial) implies
  ((self.outgoing->size <= 1) and (self.incoming->isEmpty))
```

[2] Egy végső pszeudo állapotnak nem lehetnek kimenő átmenetei

```
(self.kind = #final) implies (self.outgoing->isEmpty)
```

[3] A történet csúcsoknak legfeljebb egy kimenő átmenetük lehet

```
((self.kind = #deepHistory) or (self.kind = #shallowHistory)) implies
  (self.outgoing->size <= 1)
```

[4] Egy egyesülés csúcspontnak legalább két bejövő, és pontosan egy kimenő átmenettel kell rendelkeznie

```
(self.kind = #join) implies
  ((self.outgoing->size = 1) and (self.incoming->size >= 2))
```

[5] Egy elágazási csúcsnak legalább két kimenő, és pontosan egy bejövő átmenettel kell rendelkeznie

```
(self.kind = #fork) implies
  ((self.incoming->size = 1) and (self.outgoing->size >= 2))
```

[6] Egy ág csúcsnak egy bejövő átmenet szegmenssel, és legalább két kimenő átmenet szegmenssel, valamint őrökkel kell rendelkeznie

```
(self.kind = #branch) implies
  ((self.incoming->size = 1) and
     ((self.outgoing->size >= 2) and self.outgoing->forAll(t |
           t.guard->size = 1)))
```

ÁllapotGép (StateMachine)

[1] Egy ÁllapotGép vagy egy osztályozón belül vagy egy viselkedési tulajdonságon belül van aggregálva

```
self.context.oclIsKindOf(BehavioralFeature) or
self.context.oclIsKindOf(Classifier)
```

[2] Egy legfelső állapot mindig kompozit

```
self.top.oclIsTypeOf(CompositeState)
```

[3] Egy legfelső állapotnak nem lehetnek szülei

```
self.top.parent->isEmpty
```

[4] A legfelső állapot nem lehet egy átmenet forrása vagy célja

```
(self.top.outgoing->isEmpty) and (self.top.incoming->isEmpty)
```

[5] Nem lehetnek történeti csúcsok a legfelső állapotban

```
self.top.substate->select( oclIsTypeOf(Pseudostate) )->
   forAll (p : Pseudostate |
      not (p.kind = #shallowHistory) and not (p.kind = #deepHistory))
```

[6] Ha egy ÁllapotGép egy viselkedési tulajdonságot ír le, nem tartalmaz HívóEsemény típusú triggereket a kezdeti átmenet triggerén kívül (ld. OCL; Átmenet [8]).

```
self.context.oclIsKindOf(BehavioralFeature) implies
self.transitions->reject(
   source.oclIsKindOf(Pseudostate) and
         source.oclAsType(Pseudostate).kind= #initial).trigger->isEmpty
```

Átmenet (Transition)

[1] Egy elágazás szegmensnek nem lehetnek őrei vagy triggerei

```
self.source.oclIsKindOf(Pseudostate) implies
   ((self.source.oclAsType(Pseudostate).kind = #fork) implies
      ((self.guard->isEmpty) and (self.trigger->isEmpty)))
```

[2] Egy egyesülés szegmensnek nem lehetnek őrei vagy triggerei

```
self.target.oclIsKindOf(Pseudostate) implies
   ((self.target.oclAsType(Pseudostate).kind = #join) implies
      ((self.guard->isEmpty) and (self.trigger->isEmpty)))
```

[3] Egy elágazás szegmensnek mindig egy állapotot kell megcéloznia

```
self.source.oclIsKindOf(Pseudostate) implies
   ((self.source.oclAsType(Pseudostate).kind = #fork) implies
                        (self.target.oclIsKindOf(State)))
```

[4] Egy egyesülés szegmensnek mindig egy állapotból kell származnia

```
self.target.oclIsKindOf(Pseudostate) implies
   ((self.target.oclAsType(Pseudostate).kind = #join) implies
         (self.source.oclIsKindOf(State)))
```

[5] Egy ág szegmensnek nem lehet triggere

```
self.source.oclIsKindOf(Pseudostate) implies
   (((self.source.oclAsType(Pseudostate).kind = #branch) or
   (self.source.oclAsType(Pseudostate).kind = #deepHistory) or
   (self.source.oclAsType(Pseudostate).kind = #shallowHistory) or
   (self.source.oclAsType(Pseudostate).kind = #initial)) implies
                                          (self.trigger->isEmpty))
```

[6] Egy egyesülés szegmensnek ortogonális állapotból kell származnia

```
self.target.oclIsKindOf(Pseudostate) implies
   ((self.target.oclAsType(Pseudostate).kind = #join) implies
            (self.source.parent.isConcurrent))
```

[7] Egy elágazás szegmensnek ortogonális állapotot kell megcéloznia

```
self.source.oclIsKindOf(Pseudostate) implies
   ((self.source.oclAsType(Pseudostate).kind = #fork) implies
         (self.target.parent.isComposite))
```

[8] Egy kezdeti átmenetnek a legfelső szinten lehet triggere a <<create>> (létrehoz) sztereotípussal. Egy viselkedési tulajdonságot modellező ÁllapotGép kezdeti átmenetének van HívóEsemény triggere, mely kapcsolódik a ViselkedésiTulajdonsággal. Ezen eseteken kívül egy kezdeti átmenetnek soha sincs triggere.

```
self.source.oclIsKindOf(Pseudostate) implies
   ((self.source.oclAsType(Pseudostate).kind = #initial) implies
      (self.trigger->isEmpty or
      ((self.source.parent = self.stateMachine.top) and
               (self.trigger.stereotype.name = 'create')) or
      (self.stateMachine.context.oclIsKindOf(BehavioralFeature) and
               self.trigger.oclIsKindOf(CallEvent) and
               (self.trigger.oclAsType(CallEvent).operation =
                                          self.stateMachine.context))
      ))
self.source.oclIsKindOf(Pseudostate) implies
   ((self.source.kind = #initial) implies
      (self.trigger.isEmpty or
      ((self.source.parent = self.StateMachine.top) and
               (self.trigger.stereotype.name = 'create')) or
      (self.StateMachine.context.oclIsKindOf(BehaviouralFeature) and
               self.trigger.oclIsKindOf(CallEvent) and
               (self.trigger.operation = self.StateMachine.context))
      ))
```

.4 Szemantika

E szakasz az állapot gépek végrehajtási szemantikáját írja le. Kényelem kedvéért a szemantika leírása műveleti stílus alkalmazásával egy elméleti gép operációinak kifejezéseivel történt, mely elméleti gép egy állapot gép specifikációt implementál. Ennek az absztrakt állapot gépnek a komponensei általános esetben a következők:

- egy esemény várakozási sor, mely a bejövő esemény példányokat fogadja,
- egy diszpécser (intéző), mely kiválasztja és kiemeli a várakozási sorból az esemény példányokat a feldolgozáshoz, és
- egy esemény feldolgozó, mely az UML állapot gépek és az aktuális állapot gép specifikus, illetve általános szemantikájának megfelelően feldolgozza a részére megküldött esemény példányokat. Az elkövetkezőkben ez utóbbi komponensre “az állapot gép” -ként utalunk.

Mindez csupán hivatkozási célokat szolgál és nem jelenti, hogy az egyedi megvalósításoknak idomulniuk kell e struktúrához. Például az esemény diszpécser szerepét betöltheti néhány más objektum is, melyek egyszerűen meghívnak egy operációt egy objektumon.

Az állapot gépek dinamikus szemantikájának megértéséhez értelmezni kell az egyes metaosztályok közötti összetett kapcsolatokat. Éppenezért az állapot gép dinamukus szemantika leírásának zöme az állapot gép metaosztály kontextusa alatt található.

ÁllapotGép (StateMachine)

Szoftver kontextusban feltételezzük, hogy egy állapot gép reagál azon eseményekre, melyeket néhány külső objektum alkalmaz hozzá.

Az állapot gép esemény feldolgozása *lépésekre* van felbontva, minden lépést egy az állapot géphez irányított esemény példány idéz elő.

Az alap szemantika azt feltételezi, hogy az események feldolgozása szekvenciálisan történik, ahol minden esemény egy *run-to-completion* (RTC) (befejezésig futó) lépést stimulál. A következő külső esemény azután lesz megküldve az állapot géphez, hogy az előző RTC lépés befejeződött. Ez a feltételezés leegyszerűsíti az állapot gép átmenet funkcióját, mivel a beérkező esemény csak azután kerül feldolgozásra, miután az állapot gép egy jól definiált (stabil) *állapot konfigurációt* ért el.

Ezen szemantika gyakorlati jelentése a *vezérfonál védelem (thread protection)*, mely lehetővé teszi, hogy az állapot gép biztonságosan befejezhessen egy RTC lépést, nem áll fenn annak a veszélye, hogy egy rákövetkező esemény félbeszakítja. Ez egy vezérfonál esemény-hurokkal implementálható, mely az eseményeket sorban egymás után olvassa (aktív osztályok esetén) egy várakozási sorból (LIFO szerkezet), vagy monitorként funkcionál (passzív osztályok esetén).

Lehetséges állapot gép szemantikát definiálni úgy is, hogy az RTC lépések konkurens alkalmazását inkább egy kompozit állapot ortogonális régióihoz engedélyezzük, mint az egész állapot géphez. Ennek a definíciónak lehetővé kell tennie a szekvenciális kényszer enyhítését. Azonban az efféle szemantika igen árnyalt és nehéz implementálni. Éppenezért a dinamikus szemantika definíciója e dolgozatban arra az elvre épül, hogy egy RTC lépést az egész állapot gépre alkalmazunk. Ez a legtöbb gyakorlati célt kielégíti.

RTC feldolgozás

Ha egyszer egy esemény példány elküldésre került, ez eredményezheti egy vagy több átmenet kiváltásának engedélyezését. (Csak a megfelelő esemény típus által triggerelt átmenetek engedélyezhetők.) Alapértelmezés szerint, ha nincs engedélyezve átmenet, akkor az állapot gép az eseményt mindenféle hatás nélkül félreteszi. Egy esemény felfüggeszthető egy későbbi feldolgozáshoz, amennyiben felfüggesztett eseményként specifikálták az aktív állapotok egyikében. A felfüggesztett esemény szemantikájának leírása a következő szakaszban található.

Abban az esetben, ahol egy vagy több átmenet engedélyezett, az állapot gép kiválaszt egy részhalmazt és aktivizálja őket, és az állapotgép az egyik aktív állapot konfigurációból áttér egy új aktív állapot konfigurációba. Ezt az alap transzformációt hívjuk *lépésnek (step)*. Az állapotokat, melyek aktivizálódtak, az *átmenet kiválasztó funkció* határozza meg, ennek leírását ld. lentebb. A műveletek, melyeket az átmenet

érvényre juttatása eredményezett, előidézhetik, hogy az esemény példányok az aktuális illetve más objektumokhoz generálódjanak.

Amennyiben ezek a műveletek szinkron műveletek, akkor az átmenet *befagyasztásra* kerül egészen addig, amíg a meghívott objektum be nem fejezte futását. Minden ortogonális alsó szintű komponens legfeljebb egy átmenetet, - mint az esemény példány megküldésének eredményét - válthatja ki. Az ütköző átmenetek (ld. lentebb) nem váltják ki ugyanazt a lépést. Amikor minden ortogonális régió befejezte az átmenet végrehajtását, az esemény példány "elhasználódik" (feldolgozásra kerül) és a lépés lezáródik.

Az a sorrend, melyben a kiválasztott átmenetek aktivizálódnak, nincs definiálva. Olyan tetszőleges kereszteződésen alapszik, melyet nem definiál explicit módon az állapot gép formalizmus.

Befejező átmenetek és befejező események

A befejező átmenet egy trigger nélküli átmenet (lehet, hogy egy őr). A befejező átmenet tipikusan a forrás állapotának műveleteit zárja le.

Az állapotgép miután reagált egy esemény előfordulására, elérhet egy olyan állapot konfigurációt, ahol néhány állapotnak kimenő befejező átmenete van (átmenet konfigurációk). Az ilyen konfigurációt nem stabil konfigurációnak tekintjük.

Ebben az esetben további lépéseket kell megtenni egészen addig, amíg az állapot gép el nem ér egy stabil állapot konfigurációt (azaz több átmenet már nem engedélyezett). A befejező átmeneteket a befejező események triggerelik, melyek minden alkalommal megküldésre kerülnek az állapot géphez, valahányszor az egy állapot konfigurációval találkozik. A befejező események egy lépéssorozat formájában elküldésre kerülnek egészen addig, amíg egy stabil konfiguráció az esemény példány által bevezetett RTC lépést le nem zárta. Ennél a pontnál a vezérlés visszatér a diszpécserhez, és egy új esemény példány kerül megküldésre.

Lehetséges, hogy egy állapot gép soha nem ér el egy stabil konfigurációt. (Egy gyakorlati megoldás az ilyen esetek leküzdésére e szemantika implementációjában, hogy maximáljuk a megengedhető lépések számát, mielőtt az állapot gép elér egy stabil konfigurációt.)

Egy esemény példány érkezhet úgy is egy állapot géphez, hogy *befagyasztja* azt ugyanazon vezérlési szál egy másik objektumától származó RTC lépés közben. Ezt az esemény példányt az állapot gép ortogonális komponensei úgy is kezelhetik, hogy nem fagyasztják be az adott átmenet rendelkezésére álló időn belül.

Lépés szemantika

Egy lépés szemantikája kötetlen módon magába foglalja egy aktív, kurrens állapot konfiguráció nem ütköző átmeneteiből álló maximális halmaz végrehajtását.

(Megjegyzendő, hogy e szakasz az ***Állapot***, ***KompozitÁllapot*** és ***Átmenet*** dinamikus szemantikáján alapul.)

Átmenet kiválasztás

Az átmenet kiválasztás meghatározza, hogy az engedélyezett átmenetek mely részhalmaza lesz aktivizálva. A következő szakaszok azt a két fő szempontot tárgyalják, melyek hatással vannak az átmenet kiválasztásra: ütközések és prioritások.

ÜTKÖZÉSEK

Egy állapot gépen belül egy adott állapotban lehetséges, hogy több átmenet engedélyezett. A probléma ez esetben az, hogy ezek közül melyeket lehet egyidejűleg aktivizálni ellentmondás (ütközés) nélkül. Például, ha van két átmenet, mely egy állapottól származik, jelölje az egyiket [c1], a másikat [c2], és ha a [c1] és a [c2] egyaránt igaz, akkor csak egy átmenet aktivizálható.

Két átmenet ütközik, ha mindketten ugyanabba az állapotba lépnek, vagy pontosabban, ha annak az állapothalmaznak a csomópontja, melybe éppen léptek nem üres. Az az intuíció, hogy csak ‘konkurens‘ átmenetek aktivizálhatók egyidejűleg. E megszorítás garantálja, hogy az új aktív állapot konfiguráció, melyet az átmenetek halmazának végrehajtása eredményezett, megfelelő kialakítású lesz.

PRIORITÁSOK

A prioritások megoldják az átmenet ütközéseket, de nem mindegyiket. Az ütköző átmenetek közötti prioritások definiálásához állapot hierarchiát használunk. Definíció szerint egy átmenet kiválása egy alállapotból magasabb prioritással rendelkezik, mint egy ütköző átmenet, mely a tartalmazott állapotok bármelyikéből vált ki.

Egy átmenet prioritásának definíciója forrás állapotán alapul. Az egyesülő átmenetek a prioritásukat a legalsó forrásállapotnak megfelelően szerzik meg.

Ha t1 egy olyan átmenet, melynek forrás állapota s1, és t2 forrása s2, akkor:

- Ha s1 egyik alállapota s2 -nek, akkor t1 -nek magasabb prioritása van, mint t2 -nek.
- Ha s1 és s2 nincsenek hierarchikus kapcsolatban, akkor nincs prioritás definiálva t1 és t2 között.

(Megjegyzendő, hogy más politikák is elképzelhetők. A klasszikus állapotábrákban a prioritás fordított: a szülő állapotok magasabb prioritásokat foglalnak magukba, mint a beágyazott állapotok. Azonban az objektum kontextusban a belső állapotok specializáltabbak, mint a szüleik, ezért fölülírják őket.)

Szelektáló átmenetek

Az átmenetek azon halmaza, mely aktivizálásra kerül, a következő feltételeket elégíti ki:

- A halmazban lévő összes átmenet engedélyezett.
- Nincsenek ütközések a halmazon belül.
- Nincs átmenet a halmazon kívül, melynek magasabb prioritása van, mint a a halmazon belüli átmenetnek. Értelemszerűen tehát a magasabb prioritásúak a halmazon belül, míg az alacsonyabb priorításúak azon kívül maradnak.

E definíció nem algoritmikusan íródott, de könnyen implementálható egy "mohó" szelekciós algoritmus által, az aktív állapot konfiguráció egy nyílt átmenetével. Az aktív állapotok alulról felfelé haladnak, ahol az átmenetek az összes kiértékelt átmenettől származnak.

Ez utóbbi garantálja, hogy a prioritás szabályt senki sem szegheti meg. Az egyetlen nem triviális probléma minden szinten feloldani az ortogonális állapotokon keresztül fellépő átmenet ütközéseket. Ezt a problémát az összes olyan ortogonális állapot lezárása oldja meg, melyet egyszer egy átmeneten belül valamely komponense aktivizált. Az alulról felfelé haladás és az ortogonális állapot lezárás együtt garantál egy megfelelő szelekciós halmazt.

Felfüggesztett események

Az aktív állapot konfigurációban az összes állapot specifikálhatja a felfüggesztett események egy halmazát.

Egy esemény példány mindaddig felfüggesztett marad, amíg az ő eseményét fel nem függeszti az aktív konfiguráció. A következő RTC lépés, ahol az állapot gép elér egy konfigurációt, melyben az esemény nem felfüggesztett, az esemény példány *kész* az újra elküldésre.

Megjegyzendő, hogy a diszpécser mechanizmus felelőssége az események sorba helyezése, mivel a lépés szemantika egyszerű eseménykuldést feltételez.

Állapot (State)

A végrehajtás során egy állapot lehet *aktív* vagy *inaktív*. Egy állapot aktívvá válik, amikor belép, mint valamely átmenet eredménye, és inaktívvá válik, ha kilépett mint valamely átmenet eredménye.

Egy állapot képes kilépni és belépni mint valamely átmenet eredménye (azaz lehet önátmenet is).

Bármikor, amikor egy állapot belép, végrehajtja belépési műveletsorozatát, és bármikor, amikor kilép, végrehajtja kilépési műveletsorozatát.

KompozitÁllapot (CompositeState)

Legális állapot konfiguráció

Minden *aktív* kompozit állapotnak a végrehajtás során követnie kell a legális aktív állapot konfigurációt, figyelembe véve az alállapotait. Ez azt jelenti, hogy a következő megszorítások mindig találkozik a végrehajtás ideje alatt (kivéve az átmenet végrehajtási periódust, mely tranziens):

- Ha a kompozit állapot nem konkurens állapot, pontosan egy alállapota aktív.
- Ha a kompozit állapot konkurens, akkor az összes alállapota (régiója) aktív.

Annak érdekében, hogy elkerüljük a legális konfiguráció megszorítások megszegését a végrehajtás alatt, a belépő és kilépő kompozit állapotok dinamikus szemantikáját úgy definiálták, hogy egy jól kialakított állapot gép mindig kielégíti őket.

Egy kompozit állapot belépése

Egy nem konkurens kompozit állapot belépése

A kompozit állapotba történő belépés során a belépési műveletsorozat a szimpla állapotéhoz hasonlatosan hajtódik végre.

- *alapértelmezett belépés*: Ha az átmenet eléri a kompozit állapot szélét, akkor az alapértelmezett (kezdeti[11]) átmenet végrehajtja, hogy belépjen a kompozit állapot alállapotainak egyikébe. Megjegyzendő, hogy a kezdeti átmeneteknek mindig engedélyezetteknek kell lenniük (az ágak esetében). Egy letiltott kezdeti átmenet egy rosszul definiált végrehajtási állapot és ennek kezelése implementációs probléma.
- *explicit belépés*: Ha az átmenet "átjuttatta" az állapotot egyik alállapota felé, akkor az explicit alállapot válik aktívvá és rekurzívan követi a belépő eljárást.
- *történeti belépés*: Ha az átmenete egy kompozit állapot történeti pszeudoállapotába lép, akkor az aktív alállapot a legújabb aktív alállapot prioritásúként lesz meghatározva a belépéshez. Ha először lépett be az állapot, akkor az aktív alállapotot az átmenet határozza meg a történeti pszeudo állapotból kifelé haladva. Ha nincs ilyen átmenet specifikálva, a szituáció illegális és megoldása implementáció függő. Az aktív alállapotot az a történet határozza meg, mely az alapértelmezett belépésével együtt folyamatban van.
- *részletes történeti belépés*: hasonlít a történethez, de az aktív alállapot a részletes történeti belépést is végrehajtja (rekurzívan).

Egy konkurens kompozit állapot belépése

[11] A kezdeti átmenet egy kezdeti pszeudoállapottól származó átmenet.

Bármikor, amikor egy konkurens kompozit állapot belép, minden alállapota (a "régiók") is belép, vagy alapértelmezettként, vagy explicitként. Ha az átmenet eléri a kompozit állapot szélét, akkor minden régió, mely alapértelmezett, belép. Ha az átmenet explicit módon léptet be egy vagy több rétiót (elágazást), akkor ezek a régiók explicit módon lépnek be, a többiek alapértelmezettként.

Egy kompozit állapot kilépése

Nem konkurens állapot kilépése

Az aktív alállapot(ok) kiléptek (rekurzívan). A kilépés után az aktív alállapot, a kilépési művelet kerül végrehajtásra.

Konkurens állapot kilépése

A régiók mindegyike kilépett. Ezután a kilépési műveletek végrehajtása következik.

Pszeudoállapot (Pseudostate)

A ***Pszeudoállapot*** csomópontok családját reprezentálja az állapot gépben, melyek hozzá vannak kapcsolva az állapotokhoz és az átmenetekhez mint kompozíciós elemek, a kiegészítő szemantikát hordozzák.

Egy Pszeudoállapot a következők egyike lehet:

- A *kezdeti (initial)* egy alapértelmezett csúcspontot reprezentál mely forrásként szolgál egy egyszerű átmenethez az "alapértelmezett" állapothoz. Legfeljebb egy kezdeti csúcspontnak kell lennie egy kompozit állapotban vagy állapot gépben.

- A *részletes történet (deepHistory)* egy csúcspont, melyet egy állapot és alállapotai legújabb aktív konfigurációjának rövidített formában történő reprezentációjához használunk. Egy kompozit állapotnak legfeljebb egy történeti csúcspontja lehet. Egy történeti összekötőtől származó átmenet az alapértelmezett történeti állapothoz vezet.

- A *vázlatos történet (shallowHistory)* egy csúcspont, melyet egy állapot legújabb aktív konfigurációjának rövidített formában történő reprezentációjához használunk. Egy kompozit állapotnak legfeljebb egy vázlatos történeti csúcsa lehet. (Megjegyzendő, hogy egy állapotnak részletesTörténet és vázlatosTörténet átmenete egyaránt lehet.)

- Az *egyesülés* csúcsok több olyan átmenet szegmenst kombinálnak, melyek különböző ortogonális komponensbeli forrás csúcsokból jönnek. Az egyesülés csúcsba belépő szegmenseknek nem lehetnek őreik.

- Az *elágazás* csúcsok egy bejövő átmenetet két vagy több ortogonális cél csúcshoz kapcsolnak. Egy elágazási csúcsból kimenő szegmenseknek nem lehetnek őreik.

- Az *ág* csúcs szétválaszt egy egyedülálló szegmenst kettő vagy több, őrök által megjelölt átmenet ágra. Az őrök meghatározzák, hogy melyik ág engedélyezett. Egy előre definiált "else" -el jelölt őr legfeljebb egy ágnál definiálható.

- A *végső* egy szimpla állapotot reprezentál néhány kiegészítő szemantikával. Eltérően minden más pszeudoállapottól, ez (értelemszerűen) nem tranziens állapot. Amikor a végső állapot belépett, az ő szülő kompozit állapota befejeződik, vagy kielégíti a befejezési feltételt. Abban az esetben, amikor a végső állapot szülője a legfelső állapot, az egész állapotábra befejeződik, és ez magába foglalja az egyed "életének" befejezését is, melyet az állapotábra specifikál. Ha az állapotábra egy osztályozó viselkedését specifikálja, akkor ez magába foglalja annak a példánynak a "befejezését". Abban az esetben, amikor a végső állapot szülő állapota nem a legfelső, ez egyszerűen azt jelenti, hogy a befejező átmenetek engedélyezettek.

Egy lezáró átmenet egy kimenő nem pszeudo állapot átmenete, mely nem rendelkezik címkével (esemény vagy őr). Akkor engedélyezett, ha forrás állapota elért egy végső állapotot.

AlgépÁllapot (SubmachineState)

Az *algép állapot* egy szervezési fogalom és nem vezet be újabb viselkedési szemantikát. Az a módszer, ahogyan az algép állapot elősegíti az állapot gép szegmens újra felhasználását, hasonlít arra, ahogyan az eljárások és sablonokat használjuk a hagyományos programozási nyelvekben. Egy algép állapot elősegíti még az összetett állapot gépek felbontását egyszerűbb gépekre.

Egy algép állapot lehet olyan elgondolás is, mint egy állapot gép "szubrutin", melyben egyik gép hívja a másikat, azután "visszatér" az eredeti géphez.

Átmenetek (Transitions)

Összetett átmenetek

Általános esetben egy *átmenet* egy *összetett átmenet* része. Egy összetett átmenet azon szimpla átmenetek csoportja, melyeket az *egyesülés, elágazás* és *ág* átmenetek kapcsolnak össze. Az ág csomópontok esetében csak egy szegmens van kiválasztva mindenághoz (az őrre alapozva). A dinamikus szemantika pontosan meghatározza egy összetett átmenet végrehajtását, mely a végrehajtás kifejezésben atomi (az egyesülés, elágazás és ág pszeudoállapotok, nem állapotok).

Megjegyzendő, hogy az összetett átmenetnek legfeljebb egy triggere lehet, mivel az egyesülés, elágazás és ág pszeudoállapotoknak nem lehetnek triggerei.

Egy olyan átmenet, mely aktivizál, mindig az egyik legális állapot konfigurációból a másik legális állapot konfigurációba vezet. A kompozit átmenetből származó átmenetek ha egyszer aktivizálódtak, mindig a kompozit állapot és összetevői kilépését idézik elő.

Felső szintű (“megszakítás”) átmenetek

A kompozit állapotokból származó átmenetekre néha “felső szintű” átmenetekként, vagy “megszakításokként” utalnak. Ha aktiválásra kiválasztották őket (mint lentebb részletezve van), akkor az összes belső alállapot és azok kilépési műveleteinek végrehajtását eredményezik. Megjegyzendő azonban, hogy mivel az állapot gép szemantika RTC (leírását ld. ott), szigorúan véve ezek nem igazi megszakítások, hanem inkább általánosított vagy “csoport” átmenetek.

Engedélyezett (összetett) átmenetek

Egy átmenet *engedélyezett*, ha a következőket megtartja:

- Az átmenete összes forrás átmenetének a kurrens aktív állapot konfigurációban kell lennie. Egy befejező átmenet (trigger nélküli) megköveteli, hogy forrás állapota a lezáró állapotban legyen abban az esetben, ha egy kompozit állapot.

- A trigger illeszkedik az állapot géphez elküldött esemény példányhoz. A null triggerek bármely eseményhez illeszkednek meghatározott befejező eseményben. Egy specializált esemény egy általánosított esemény alapuló triggerrel illeszkedik.

- Létezik az átmenet szegmenseknek egy útvonala a forrás állapotoktól a cél állapotokba, mely mentén az összes őr feltételei kielégítettek (az őr nélküli átmenet mindig kielégített). Ha több, mint egy ilyen útvonal lehetséges, akkor csak egy a kiválasztott (nem determinisztikusan).

Megjegyzendő, hogy az őrök kiértékelése megelőzi bármely - az átmenettel összefüggésben álló - művelet meghívását.

Mivel az őrök nem értelmezettek, kiértékelésük olyan kifejezéseket foglalhatnak magukba, melyek mellékhatásokat idéznek elő. A mellékhatásokat okozó őrök rossz gyakorlatnak tekintendők, mivel kiértékelési stratégiájuk a sorrend szempontjából nem definiált és az implementáció egy funkcióját képezi.

(Összetett) átmenet végrehajtás

Az átmenet végrehajtás szemantikát úgy definiálták, hogy az eredmény állapot konfiguráció mindig legális. E szabály különösen fontos, amikor átmenetekkel, konkurens állapotok belépési/kilépési határaival foglalkozunk.

LCA, fő forrás és fő cél

Minden összetett átmenet egy (kompozit) állapot kilépését és más kompozit állapot megfelelő belépését idézi elő. E két állapotot úgy jelölték ki, mint az átmenet ***fő forrását*** és ***fő célját***.

Egy átmenet legkisebb közös ős (LCA)[12] állapota az a legalsó állapot, mely az összetett átmenet összes explicit forrás állapotát és explicit cél állapotát tartalmazza. Ág szegmensek esetében csak a kiválasztott útvonallal kapcsolatban álló állapotok lesznek figyelembe véve explicit célokként ("halott" ágak nem lesznek figyelembe véve).

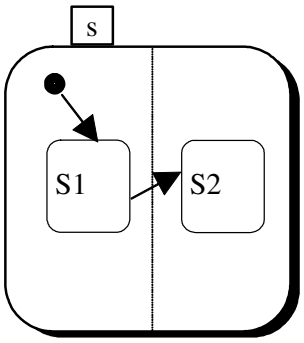
A *fő forrás* az *LCA* egy közvetlen alállapota, mely tartalmazza az explicit forrásokat. A *fő cél* az *LCA* egy alállapota, mely tartalmazza az explicit célokat.

Példák:

1. Az általános szimpla eset: Egy átmenet *t* két szimpla állapot *s1* és *s2* között egy kompozit állapotban *s*.

   Itt *LCA* (t) *s*, a fő forrás *s1* és a fő cél *s2*.

2. Egy elvontabb eset: Egy struktúrálatlan átmenet az egyik régióból a másikba



19. ábra: struktúrálatlan átmenet az egyik régióból a másikba

Itt *LCA*(t) *s* szülője, a fő forrás *s1* , és a fő cél *s2*

Átmenet végrehajtási szekvencia

Amennyiben egyszer egy átmenet engedélyezett és aktiválásra ki van választva, a következő lépéseket kell rendre végrehajtani:

A fő forrás állapot megfelelően kilép.

A műveletek végrehajtódnak a következő lineáris sorrendben, az átmenet szegmensei mentén: A "közelebbi" a forrás állapothoz való művelet, a korábbi kerül végrehajtásra.

A fő cél állapot megfelelően belép.

[12] Angol rövidítése: LCA = Least Common Ancestor, a továbbiakban e rövidítéssel hivatkozunk rá.

.5 Szabványos elemek

A Állapot Gépek csomaghoz előre definiált sztereotípusok, megszorítások és csatolt értékek felsorolása a következő táblázatban; definíciójuk pedig a *Szabványos Elemek* függelékben látható

Modell Elemek	Sztereotípusok	Megszorítások	Csatolt Értékek
Esemény	«create» (létrehoz) «destroy» (lerombol)		

Állapot Gépek - Szabványos Elemek

.6 Megjegyzések

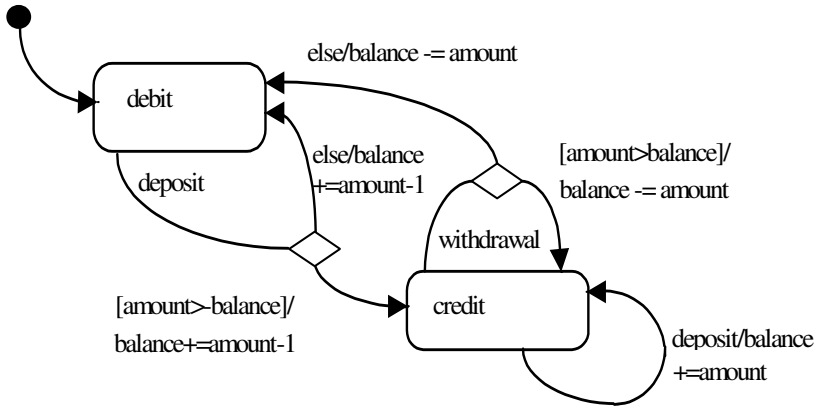
Példa: Az osztály viselkedés modellezése

Egy szoftverben, - mely implementálása nem más, mint egy állapot modellezéssel végrehajtott tervezés eredménye - az állapot gép voltaképpen vagy látható, vagy nem látható a (generált vagy kézzel írott) forráskódon belül. Nem lesz látható az állapot gép abban az esetben, ha létezik néhány futásidejű rendszer, mely támogatja az állapot gép viselkedést. Általánosabb esetben azonban a forráskód tartalmaz olyan speciális utasításokat, melyek implementálják az állapot gép viselkedését.

Egy C++ példát mutat be a következő forráskód részlet:

```
class bankSzamla
private:
int balance; // egyenleg
public:
void deposit(amount) // befizetés (összeg)
{
   if (balance > 0)     balance = balance + amount; // nincs terhelés
   else
   balance = balance + amount – 1; // $1 terhelés a tranzakcióhoz
}
void withdrawal (amount) { // összeg kivétele
if (balance > 0) balance = balance – amount;
}
}
```

A fenti programsorokban jól látható, hogy az osztálynak egy - a *balance (=egyenleg)* attributum által kinyilvánított - absztrakt állapota van, mely vezérli az osztály viselkedését. Ezt modellezi az állapot gép a 20. ábrán:



(debit=terhelés; credit=hitel)

20. ábra: Állapot Gép az osztály viselkedés modellezéséhez

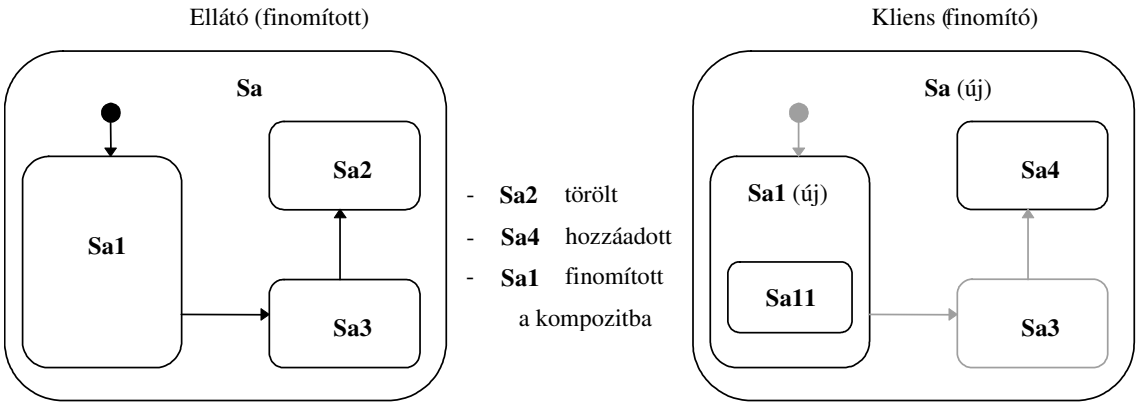
Mivel az állapot gépek az általánosítható elemek viselkedéseit írják le, elsősorban az osztályok és állapot gépek finomítása használatos a megfelelő állapot gépek közötti kapcsolatok megragadására.

Maga a finomítási mechanizmus a Kiegészítő Elemek csomag része, és általános finomítási kapcsolatokat definiál tetszőleges modell összetevők között.

Példa: Állapot gép finomítás

Mivel az állapot gépek az általánosítható elemek viselkedéseit írják le, elsősorban az osztályok és állapot gépek finomítása használatos a megfelelő állapot gépek közötti kapcsolatok megragadására. A finomítási kapcsolatok kezelését a kiegészítő elemek csomagban definiált finomítási metaosztály segíti. Az állapot gépek a finomítást három különböző leképezésben használja, melyeket a finomítási osztály leképezési attributuma specifikál. E leképezések a finomítás, a helyettesítés és a törlés.

Az állapot gépek finomításának illusztrálására nézzük a következő példát, ahol egy állapot gép csatlakozik egy “Ellátó” -val jelölt osztályhoz, ezt egy másik állapot gép finomítja, mely egy “Kliens” -sel jelölt osztályhoz csatlakozik.



21. ábra: Állapot Gép finomítási példa

A fenti példában a kliens állapot (**Sa**(új)) az alosztályban helyettesíti a szimpla alállapotot (**Sa1**) egy kompozit alállapot (**Sa1**(új)) által. Ennek az új kompozit alállapotnak van egy komponens alállapota (**Sa11**). Továbbá az **Sa1** új verziója törli az **Sa2** -t és egy új alállapotot, az **Sa4** -et adja hozzá. Az **Sa3** öröklődött és ezért közös az **Sa** mindkét verziójára. Az érthetőség kedvéért szürke színt használtunk annak a komponensnek az azonosítására, mely az eredetiből öröklődött. (E fentiek az illusztrálást szolgálják, nem fejeznek ki jelölésmódra vonatkozó ajánlást.)

Fontos megjegyezni, hogy az itt definiált állapot gép finomítás nem specifikál vagy részesít előnyben semmilyen finomítási politikát. Ehelyett egyszerűen egy rugalmas mechanizmust biztosít, mely lehetővé teszi az altípus kezelést (viselkedési kompatibilitás), öröklődést (implementációs újra felhasználás) vagy az általános finomítási politikákat.

A potenciálisan haszos politikák egy vázlatos leírását adjuk itt, mely politikák jól implementálhatók az állapot gép finomítási mechanizmussal. E politikákat a szabványos sztereotípusok (pl. <<subtype>> (altípus), vagy <<inherits>> (örököl)) csatlakoztatásával kell feltüntetni az állapot gépek közötti kapcsolat finomításához.

Altípus kezelés

Az altípus kezeléshez kidolgozott finomítási politika azon alapon nyugszik, hogy az altípus biztosítja a típuson alkalmazó események/operációk elő/utó feltétel kapcsolatait, melyeket az állapot gép specifikált. Az elő/utófeltételeket az állapotok; a kapcsolatokat pedig az átmenetek realizálják. Az elő/utófeltételek megőrzése garantálja a helyettesíthetőségi elvet.

Az állapotok és átmenetek csak hozzáadódnak, nem törlődnek. A finomítás interpretációját a következő felsorolás mutatja:

- Egy finomított Állapotnak ugyanazok a kimenő átmenetei vannak, de felvehet még ezekhez további különböző bejövő átmenetet is. Több alállapota lehet, és hamisról igazra változtathatja a konkurencia tulajdonságát.

- Egy finomított Átmenet átmehet egy új cél állapotba, mely az alap osztályban specifikált állapot alállapota. Ez garantálja az alap osztály által specifikált utófeltételt.

- Egy finomított őrnek ugyanaz az őr feltétele, de hozzáadhat bizonyos diszjunkciókat. Ez azt garantálja, hogy az előfeltételek inkább gyengülnek, mint erősödnek.

- Egy finomított MűveletSorozat ugyanazokat a műveleteket tartalmazza (és ugyanabban a sorrendben), de lehetnek további műveletei is. Az újabb műveletek nem hátráltathatják az átmenet cél állapota által reprezentált eredeti (változatlan) műveleteket.

(Szigorú) Öröklődés

E politika alapvető értelme inkább az implementáció újra felhasználásának előmozdítása, mint a viselkedés tárolása. Mivel a legtöbb implementációs környezet hasznosítja a szigorú öröklődést (azaz a tulajdonságok megváltoztathatók, hozzáadhatók, de nem törölhetők), az öröklődési politika ezt a vonalat követi azáltal, hogy tiltja azt a finomítást, mely nem szigorú öröklődéshez vezet, ha egyszer az állapot gépet implementálták.

Állapotok és átmenetek hozzáadhatók. A finomítás interpretációja a következők szerint történt:

- Egy finomított Állapot ugyanazon bejövő átmenetek némelyikét birtokolja (azaz néhányat eldob, néhányat hozzáad), ellenben a kimenő átmeneteknek egy nagyobb, ill. bővebb halmazával rendelkezik. További alállapotai lehetnek és megváltoztathatja a konkurencia attributumát.

- Egy finomított Átmenet átmehet egy új cél állapotba, de ugyanazzal a forrással kell rendelkeznie.

- Egy finomított őr az eredetitől különböző őr feltétellel rendelkezhet

- Egy finomított MűveletSorozat ugyanazokat a műveleteket tartalmazza (és ugyanabban a sorrendben), és lehetnek további műveletei is.

Általános Finomítás

Ebben a legáltalánosabb esetben az állapotok és az átmenetek hozzáadhatók és törölhetők (azaz “null” finomítások). A finomítás megszorítások nélkül értelmezett, azaz a finomított állapot gép elem és finomító elem tulajdonságaira és kapcsolataira vonatkozólag nincsenek formális követelmények:

- Egy finomított Állapotnak különböző kimenő és bejövő átmenetei lehetnek (azaz eldobhatja az összeset, ill. felvehet néhányat)

- Egy finomított Átmenet különböző forrásokból érkezhet és egy új cél állapotba mehet

- Egy finomított őr az eredetitől különböző őr feltétellel rendelkezhet

- Egy finomított MűveletSorozatnak nem szükséges ugyanazokat a műveleteket tartalmaznia (vagy megváltoztathatja sorrendjüket), és lehetnek további műveletei is.

A kompozit állapot finomítása a fenti példában az általános finomítás illusztrációja.

Meg kell itt jegyezni, hogy ha egy típusnak több szupertípus kapcsolata van a strukturális modellben, akkor a típushoz rendelt alapértelmezett állapot gép a szupertípusának összes állapot gépéből, mint ortogonális állapot gép régiókból áll. Ez explicit módon felülbírálható a finomításon keresztül, amennyiben szükséges.

Klasszikus állapotábrák

A fő különbséget a klasszikus (Harel) állapotábra és az objektum állapot gépek között az állapot gép külső kontextusa eredményezi. Az objektum állapot gépek elsősorban egy típus viselkedésének reprezentálására hivatottak. A klasszikus állapotábra a folyamatok viselkedéseit specifikálja. A következő lista a fenti magyarázatból következő különbségeket sorolja fel:

- Az események inkább hordozó paraméterek, mint primitív jelek

- A hívó események (operáció triggerek) a típusok viselkedésének modellezésére hivatottak

- Az események összetalálkozása nem támogatott, és a szemantika egyszeres eseményküldést vesz figyelembe, hogy a típus kontextushoz való illeszkedés jobb legyen, mint a vele szembeálló általános rendszer kontextussal.

- A klasszikus állapotábrák az előre definiált műveletek, feltételek és események egy gondosan kimunkált halmazával rendelkezik, mely halmazelemeket nem biztosítják az objektum állapot gépek.

- Az operációk nincsenek körbeküldve, de irányíthatók egy objektum halmaz felé.

- A tevékenységek (folyamatok) elképzelés nem létezik az objektum állapot gépekben. Éppenezért az összes előre definiált műveletek és események, melyek a tevékenységekkel foglalkoznak, nem támogatottak, csakúgy mint az állapotok és tevékenységek közötti kapcsolatok.

- Az átmenet kompozíciók gyakorlati okokból korlátozottak. A klasszikus állapotábrákban a pszeudo állapotok, szimpla átmenetek, őrök és címkék bármely kompozíciója engedélyezett.

- Az objektum állapot gép támogatja az állapot gépek közötti szinkron kommunikáció fogalmát.

- Az átmeneteken elvégzendő műveletek megadott sorrendjükben hajtódnak végre.

- A klasszikus állapotábrák a nullidő feltételezésen alapulnak. E feltételezés azt jelenti, hogy az átmenetek végrehajtása null (azaz elhanyagolható) időt vesz igénybe. Az egész rendszer végrehajtása a szinkron lépéseken alapul, ahol minden lépés egy új eseményeket generál, melyek a következő lépésnél lesznek feldolgozva. Az OO állapotgépekben e feltételezéseket elengedik ill. helyettesítik e szoftver végrehajtási modellel, mely a végrehajtási szálon alapul és melynél a műveletek végrehajtása időt vesz igénybe. (Ez a valós idejű rendszerek - mint amilyen egy MIDI - kommunikációs protokollon alapuló zenei rendszer - modellezésénél rendkívül fontos.)

.7 Aktivitás modellek

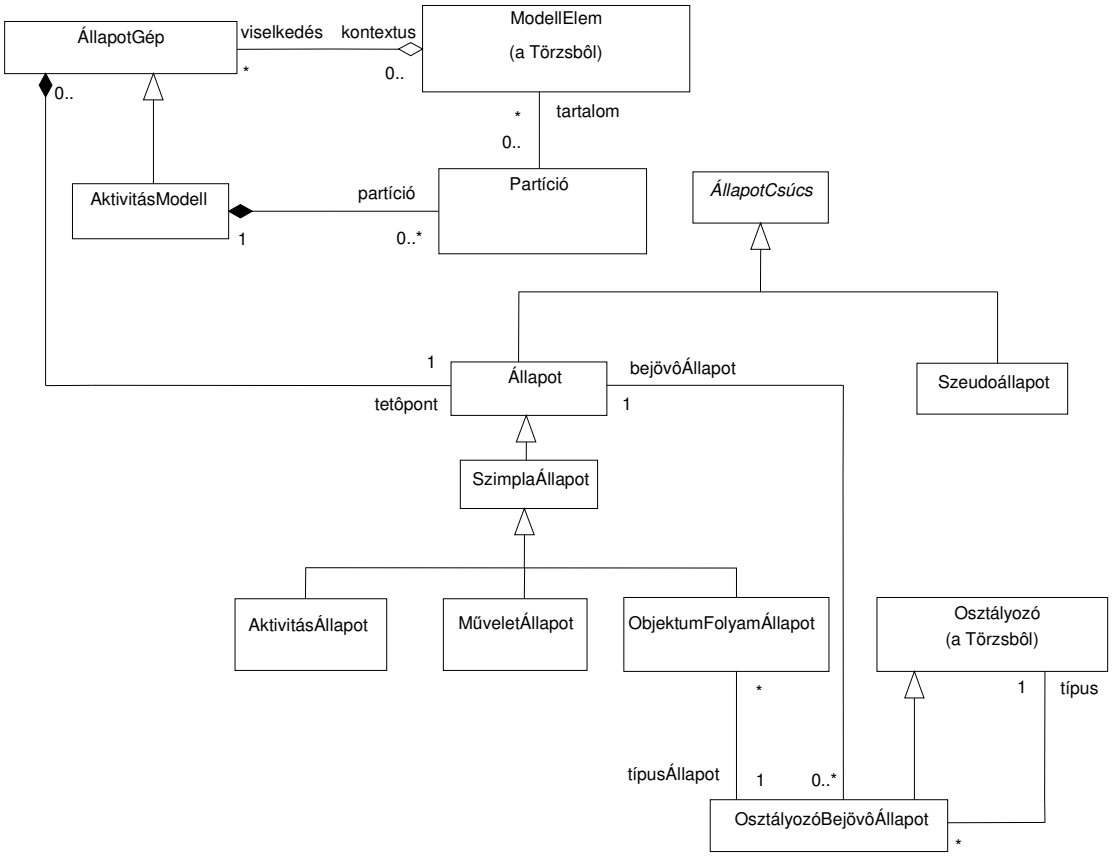
.1 Áttekintés

Az aktivitás modellek az Állapot Gépek csomag egy kiterjesztett nézetét definiálják. Az állapot gépek és az aktivitás modellek alapjában véve egyaránt állapot átmenet rendszerek és több metamodell elemet osztanak meg. E szakasz az Állapot Gép csomag azon fogalmait írja le, melyek az aktivitás modellre nézve specifikusak. Itt kell megjegyezni, hogy az aktivitás modellek kiterjesztés igen kevés saját szemantikával rendelkezik - az Állapot Gép csomag kontextusában kell értelmezni, beleértve az Alap csomag és az Általános Viselkedés csomaggal való függőségeket is.

Egy aktivitás modell az állapot gép modell speciális esete, mely egy vagy több osztályozót magukba foglaló folyamatok modellezésére használatos. A legtöbb állapot egy ilyen modellben művelete állapot, mely atomi műveletet reprezentál, azaz állapotok, melyek műveleteket hívnak meg, majd várakoznak. Az átmeneteket a művelet állapotokba események triggerelik, melyek lehetnek 1) az előző művelet állapot befejezése; 2) egy bizonyos állapotban lévő objektum rendelkezésre állása; 3) egy jel előfordulása; vagy 4) valamely feltétel kielégítése. Az alap állapot gép fogalmakat kiegészítő altípus-halmaz definiálása által az aktivitás modellek megfelelő kialakítása formálisan definiálható és utólag leképezhető az állapot gépek dinamikus szemantikájára. Ezen kívül az aktivitás specifikus altípusok kiküszöbölik a félreérthetőségeket, melyek egyébként az aktivitás modellek különböző eszközök közötti cseréjekor óhatatlanul felmerülnek.

.2 Absztrakt szintaxis

Az aktivitás modellek absztrakt szintaxisának grafikus szemléltetése a 22. ábrán látható:



22. ábra: Aktivitás Modellek

A következő metaosztályokat használjuk az aktivitás modellek definiálásához:

AktivitásModell (ActivityModel)

Egy *aktivitás modell* az állapot gép speciális esete, mely egy számítási folyamatot definiál a vezérőfolyam és objektum-folyam szempontjából az őt alkotó összetevő műveletek között. Nem terjeszti ki az állapot gépek szemantikáját, hanem rövidített formákat definiál, melyek alkalmasak a számítási folyamatok modellezésére.

Az elsődleges alap az ***AktivitásModell***ekhez leírni egy aktivitás vagy folyamat állapot modelljét, beleértve egy vagy több ***Osztályozó***t. Az aktivitás modellek hozzákapcsolhatók ***Csomagokhoz***, ***Osztályozó***khoz (beleértve a ***HasználatiEset***eket) és ***ViselkedésiTulajdonság***. A legtöbb ***Állapot*** egy aktivitás modellben ***MűveletÁllapot***, azaz olyan állapotok, melyekben egy művelet végrehajtásra kerül, tipikusan az exekutív operációk. Mint bármely állapot gépben, ha egy kimenő átmenetet nem explicit módon triggerelt (aktivizált) egy esemény, akkor az általa tartalmazott műveletek végrehajtása fogja implicit módon triggerelni. Egy ***AktivitásÁllapot*** strukturált altevékenységet reprezentál, mely valamilyen időtartammal rendelkezik és belsőleg műveletek

halmazából áll. Azaz egy AktivitásÁllapot egy "hierarchikus művelet" egy beágyazott aktivitás almodellel, mely végül is egyedi műveletekre bomlik fel.

Közönséges "várakozó állapotokat" lehet felvenni a modell szituációkhoz, melyekben a számítás egy külső esemény bekövetkeztére vár. Ágak, elágazások és egyesülések szintén felvehetők a modell döntésekhez és a konkurens aktivitásokhoz (tevékenységekhez).

Az ***AktivitásModell***ek magukba foglalják a ***Partíció*** fogalmát az állapotok szervezéséhez a különböző szempontoknak megfelelően, mint ahogyan egy valós világbeli szervezet is felelős az általa elvégzett tevékenységekért.

Az aktivitás modellezés jól alkalmazható az üzleti folyamatok tervezéséhez használt szervezeti modellezés kontextusában, valamint a munkafolyamat modellezése során. E kontextusban az események gyakran a rendszerből nézve a "külvilágtól" származnak (azaz "ügyfél általi hívások"). Az aktivitás modellek kiválóan alkalmazhatók még rendszer modellezési célokra operációk és rendszer szintű folyamatok dinamikájának specifikálásához amikor nem szükséges egy teljes kölcsönhatás modell.

Asszociációk (Associations)

partíció (partition) Azon ***Partíciók*** halmaza, melyek a modell modell elemei közül tartalmaz néhányat.

MűveletÁllapot (ActionState)

Egy *művelet állapot* egy atomi művelet végrehajtását reprezentálja, tipikusan egy operáció meghívását.

Egy ***MűveletÁllapot*** egy ***SzimplaÁllapot*** egy belépési művelettel, melynek csak a kilépési ***Átmenetét*** triggereli a belépési művelet végrehajtás befejezésének implicit eseménye. Az állapot éppenezért megfelel magának a belépési műveletnek a végrehajtásával és a kimenő ***Átmenet*** aktiválódik amint a művelet befejezte a végrehajtását.

Egy ***MűveletÁllapot*** végrehajthat egynél több ***Műveletet***, mint az ő belépési ***MűveletSorozatának*** részét. Egy ***MűveletÁllapot***nak nem lehet kimeneti átmenete, belső átmenete, vagy külső átmenete, melye(ket) bármely más triggerel, mint az implicit művelet befejezési esemény.

Asszociációk (Associations)

belépés (entry) (Az ***Állapot***tól öröklődik.) Specifikálja a meghívott műveleteket.

AktivitásÁllapot (ActivityState)

Egy *aktivitás állapot* egy nem atomi lépéssorozat végrehajtását reprezentálja, melynek valamilyen időtartama is van (azaz belső felépítését tekintve műveletek halmazából áll és esemény(ek)re való várakozás is lehetséges). Azaz egy aktivitás állapot egy "hierarchikus művelet", ahol egy kapcsolódó al-tevékenység modell végrehajtódik.

Egy ***AktivitásÁllapot*** egy ***AlgépÁllapot***, mely végrehajtja a beágyazott aktivitás modellt. Amikor egy input átmenet az ***AktivitásÁllapot***hoz, aktiválódik, a végrehajtás megkezdődik a beágyazott ***AktivitásModell*** kezdeti állapotával. Egy ***AktivitásÁllapot*** kimenő ***Átmenete*** engedélyezett amikor a beágyazott ***AktivitásModell*** végső állapotát elérte (azaz amikor befejeződött a végrehajtása).

Asszociációk (Associations)

algép (submachine) (Az ***AlgépÁllapot***tól öröklődik.) Kijelöl egy aktivitás modellt, mely fogalmilag az aktivitás állapotba van beágyazva. Az aktivitás állapot fogalmilag egyenértékű egy ***KompozitÁllapottal*** mely tartalmát a beágyazott ***AktivitásModell*** képezi. A beágyazott aktivitás modellnek kell lennie kezdeti- és végállapotának.

OsztályozóBejövőÁllapot (ClassifierInState)

Egy *osztályozó bejövő állapot* egy adott osztályozó példányait jellemzi egy bizonyos állapothoz. Egy aktivitás modellben az *osztályozó bejövő állapot* lehet input és/vagy output egy művelethez egy objektum folyam állapoton keresztül.

Az ***OsztályozóBejövőÁllapot*** az ***Osztályozó*** altípusa és felhasználható statikus struktúrális modellekhez és együttműködésekhez; azaz használható olyan asszociációk bemutatásához, melyek melyek csak egy adott állaotban levő osztály objektumainál relevánsak.

Asszociációk (Associations)

típus (type) Kijelöl egy ***Osztályozót***, mely jellemzi a példányokat.

bejövőÁllapot (inState) Kijelöl egy ***Állapotot***, mely jellemzi a példányokat. Az állapotnak a megfelelő ***Osztályozó*** egy érvényes állapotának kell lennie.

ObjektumFolyamÁllapot (ObjectFlowState)

Egy *objektum folyam állapot* műveletek között definiál objektum folyamot egy aktivitás modellen belül. Egy adott állapotban lévő osztályozó példány rendelkezésre állását jelenti, rendszerint egy operáció eredményeként.

Az objektum használata egy őt követő ***MűveletÁllapotban*** modellezhető az ***ObjektumFolyamÁllapot*** output átmenetének csatlakoztatása által, mint input átmenet a ***MűveletÁllapotba***. Általában minden művelet az objektumot egy különböző állapotba helyezi, mely eltérő ***ObjektumFolyamÁllapot***ként modellezhető.

Asszociációk (Associations)

típusÁllapot (typeState) Kijelöli az objektum osztályát (vagy más osztályozóját) és állapotát.

Partíció (Partition)

A *partíció* mechanizmus az aktivitás modell állapotait osztja csoportokba. A partíciók gyakran megfelelnek egy üzleti modellbeli szervezeti egységeknek. Jól felhasználhatók karakterisztikák vagy erőforrások allokálására egy aktivitás modell állapotai közé.

Asszociációk (Associations)

tartalom (contents) Azon állapotokat specifikálja, melyek a partícióhoz tartoznak. Nem szükséges beágyazott régió létrehozása.

Itt szükséges megjegyezni, hogy a ***Partíciók*** nincsenek hatással a modell dinamikus szemantikájára, de segítenek tulajdonságokat és műveleteket allokálni különböző célokhoz.

PszeudoÁllapot (PseudoState)

A *pszeudo állapot* a különböző típusú csomópontok egy absztrakciója egy állapot gép ábrán belül mely átmeneti pontokként funkcionál az egyik állapotból a másikba történő átmenetekben, mint amilyen az elágazás.

A *végső* ***PszeudoÁllapot***ok a hierarchikus tevékenységek modellezéséhez használjuk. Egy *végső* ***PszeudoÁllapot***ba való átmenetet egy ***AktivitásModell***ben egy al-***AktivitásModell*** befejezésének jelzésére használhatunk úgy, hogy a végrehajtás a szuperállapot szinten folytatódik (azaz az kimenő szuperállapot átmenetek aktiválódnak). Egy beágyazott aktivitás modellnek egyaránt kell rendelkeznie kezdeti- és végállapottal (vagy állapotokkal).

.3 Jól képzett szabályok

AktivitásModell (ActivityModel)

[1] Egy AktivitásModell egy (I.)*Csomag*, vagy (II.) *Osztályozó* (beleértve a *HasználatiEset*et), vagy (III.) *ViselkedésiTulajdonság* dinamikáját specifikálja.

```
(self.context.oclIsTypeOf(Package)    xor
 self.context.oclIsKindOf(Classifier) xor
 self.context.oclIsKindOf(BehavioralFeature))
```

[2] Egy olyan AktivitásModell, mely egy ViselkedésiTulajdonság dinamikáját specifikálja, vagy amely beágyazott, pontosan egy kezdeti állapottal rendelkezik, reprezentálva a ViselkedésiTulajdonság vagy altevékenység meghívását.

MűveletÁllapot (ActionState)

[1] Egy MűveletÁllapot pontosan egy kimenő Átmenettel rendelkezik.

```
self.outgoing->size = 1
```

[2] Egy MűveletÁllapotnak van egy nem üres belépési MűveletSorozata.

```
self.entry.action->size > 0
```

[3] Egy MűveletÁllapotnak nincs belső Átmenete vagy Kilépési MűveletSorozata.

```
self.internalTransition->size = 0 and self.exit->size = 0
```

ObjektumFolyamÁllapot

[1] Az ObjektumFolyamÁllapot OsztályozóBejövőÁllapota egy Operációhoz szolgáló input Paraméter típusa. A szóbanforgó Operáció a MűveletÁllapotokon belül kerül meghívásra, melynek ObjektumFolyamÁllapota van egy bejövő Átmeneten.

```
self.outgoing.target->select(oclIsTypeOf(ActionState)).
      invoked.parameter->select(
            kind = #in or kind = #inout).type->includes(self.typeState.type)
```

[2] Az ObjektumFolyamÁllapot OsztályozóBejövőÁllapota egy Operáció output Paraméterének a típusa. A szóbanforgó Operáció a MűveletÁllapotokon belül kerül meghívásra, melynek ObjektumFolyamÁllapota van egy bejövő Átmeneten.

```
self.incoming.source->select(oclIsTypeOf(ActionState)).
      invoked.parameter->select(
            kind = #out or kind = #inout or kind = #return).
                  type->includes(self.typeState.type)
```

PszeudoÁllapot (PseudoState)

[1] Az AktivitásModellben, azon Átmeneteknek, melyek beérkeznek (vagy kiindulnak) az *egyesülés* és *elágazás* PszeudoÁllapotokba, rendelkezniük kell forrásokkal (célokkal). Azaz az *egyesülések* és *elágazások* a felhaszálás tekintetében szintaktikusan nincsenek korlátozva a KompozitÁllapotokkal való kombinációban való felhasználásra, mint az ÁllapotGépek esetében.

```
self.stateMachine.oclIsTypeOf(ActivityModel) implies
  ((self.kind = #join or self.kind = #fork) implies
        (self.incoming->forAll(source.oclIsKindOf(SimpleState) or
                                    source.oclIsTypeOf(PseudoState)) and
        (self.outgoing->forAll(source.oclIsKindOf(SimpleState) or
                           source.oclIsTypeOf(PseudoState)))))
```

[2] Minden elágazást elhagyó útvonalnak végsősoron újra egyesülnie kell egy soron következő egyesülési pontnál vagy pontoknál. Továbbá, ha az egyesüléseknek több szintje létezik, ezeknek jól beágyazottaknak kell lenniük. Éppenezért egy aktivitás modell konkurencia struktúrája tulajdonképpen egyenlően korlátozó, mint egy közönséges állapot gépé, habár a kompozit állapotoknak nem kell expliciteknek lenniük.

.4 Szemantika

AktivitásModell (ActivityModel)

Az aktivitás modellek dinamikus szemantikája az állapot gépek szemszögéből fejezhető ki. Ez azt jelenti, hogy a tevékenységek (aktivitások) folyamat struktúrájának egyenértékűnek kell lennie az ortogonális régiókkal (a kompozit állapotokban). Azaz a párhuzamos útvonalakat keresztező átmenetek (vagy vezérlési szálak) nem engedélyezettek. Így egy tevékenység (aktivitás) specifikáció, mely "korlátozás nélküli párhuzamosságot" tartalmaz, - mint ahogy az általános aktivitás modellekben használatos, - befejezetlennek tekintendők az UML szempontjából.

Minden esemény, mely egy adott állapot szempontjából nem releváns, fel kell függeszteni úgy, hogy felhasználhatók legyenek, amikor relevánssá válnak. Ezt az állapot gépek általános felfüggesztési mechanizmusa segíti elő.

MűveletÁllapot (ActionState)

Amint egy MűveletÁllapot bejövő átmenete aktiválódik, a belépési művelete megkezdi végrehajtását. Ha egyszer egy belépési művelet befejezte a végrehajtását, a művelet befejezettnek tekinthető. Ennélfogva - formálisan - egy aktivált művelet állapot azt jelenti, hogy egy művelet végrehajtása folyamatban van. Amikor a művelet befejeződött, akkor a kimenő átmenet (vagy egy szimpla átmenet, vagy egy "elágazás") engedélyezett.

ObjektumFolyamÁllapot (ObjectFlowState)

Egy ObjektumFolyamÁllapot aktiválása azt jelenti, hogy a kapcsolódó Osztályozó egy példánya rendelkezésre áll egy meghatározott állapotban (azaz állapot változás fordult elő, mint az előző operáció eredménye). Ez megengedheti egy soron következő műveletnek, hogy inputként igényelje a példányt. A művelet végrehajtása felhasználja az értéket. Amennyiben az ***ObjektumFolyamÁllapot*** egyesülés pszeudoállapothoz vezet, akkor az ObjektumFolyamÁllapot aktivált marad egészen addig, amíg az egyesülés más elődje be nem fejeződött.

A MűveletÁllapot meghívása általában az objektum állapot változását eredményezi, ez pedig egy új ObjektumFolyamÁllapotot idéz elő.

.5 Megjegyzések

Az objektum-folyam állapotok az aktivitás modellekben a folyamat modellek általános *adatfolyam* aspektusának specializációja. Az objektum-folyam aktivitás modellek három területen terjesztik ki a szabványos adatfolyam kapcsolatok szemantikáját:

- Az operációk a művelet állapotokban az aktivitás modelleken belül az osztályok vagy a típusok operációi. Nem működnek hierarchikus "funkciók" egy adatfolyamon.

- Az objektum folyam állapotok "tartalma" típusos. Nem struktúrálatlan adat definíciók.

- Az objektum állapota, mely objektum az operációk között inputként és outputként áramlik, explicit módon van definiálva. Az objektum folyam állapotok - értelemszerűen - nem állapot nélküli passzív adatdefiníciók, mint az adattárolók.

5. IV. RÉSZ: ÁLTALÁNOS MECHANIZMUSOK

A negyedik rész a modellek általános alkalmazhatóságának mechanizmusait definiálja. Az UML e verziója egy általános mechanizmus csomagot tartalmaz, a Modell Menedzsmentet. A Modell Menedzsment csomag azt határozza meg, hogy a modell elemek hogyan szerveződnek modellekbe, csomagokba és rendszerekbe.

Tartalom

1 Modell Menedzsment Csomag

.1 Áttekintés

A Modell Menedzsment csomag a Viselkedési Elemek csomag egy alcsomagja. ***Modell, Csomag*** és ***Alrendszer*** elemeket definiál, melyek főként egységek csoportosítására szolgál más ***ModellElem***ekhez. A csomag az UML Alap- és Általános Viselkedés csomagjában definiált konstrukciókat használja.

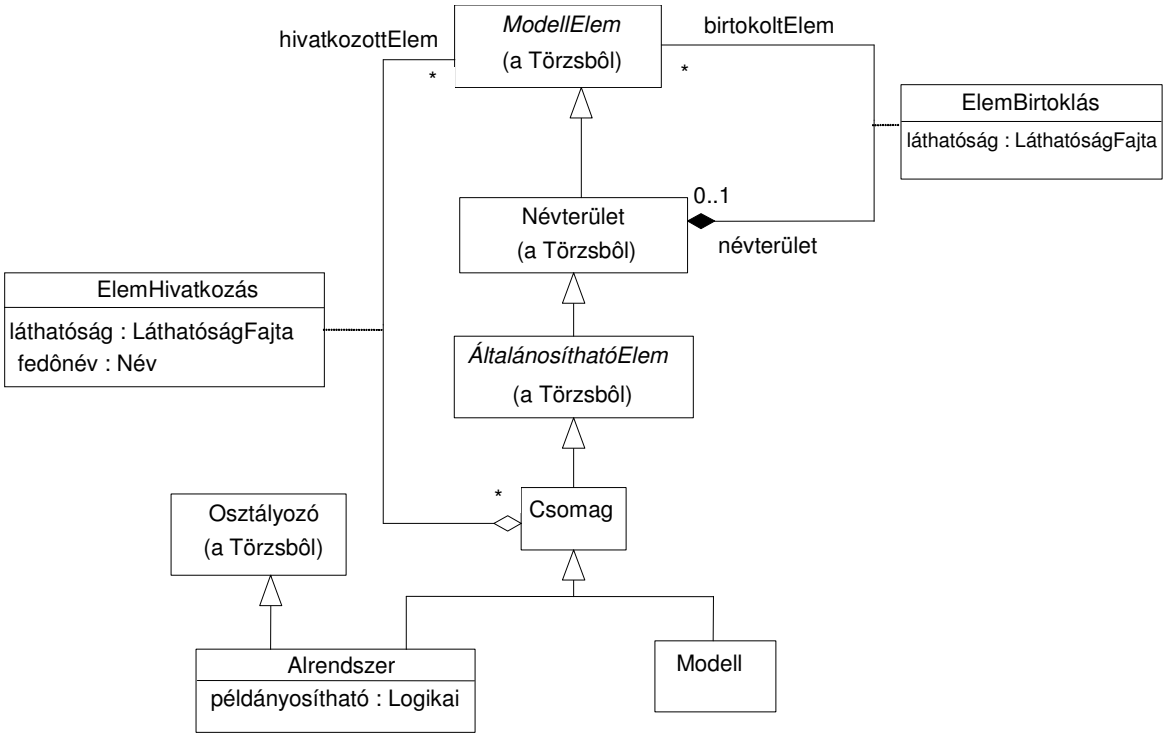
A ***Csomagok*** egy ***Modell***en belül haszálatosak a ***ModellElem***ek csoportosítására. Egy ***Alrendszer*** a ***Csomag*** speciális fajtája az ***Alrendszer***beli ***ModellElem*** által felkínált viselkedés kiegészítő specifikációjával.

E szakaszban a *modellezett rendszer* azt a fizikai egyedet jelenti, melyet az UML -el modellezünk, azaz a kifejezés nem a modellezési nyelv konstrukcióinak egyike. Jelenthet egy számítógépes rendszert, mint amilyen egy *helyfoglalási rendszer, egy banki rendszer, vevő- és szállító folyószámla rendszer, vagy egy telefon forgalmi rendszer.* Leírhat ezen kívül üzleti folyamatokat is, mint amilyenek a *kereskedelmi folyamat*, egy *mérésügyi hitelesítési eljárás,* vagy egy *fejlesztési folyamat*. Egy analógia lehet a házak konstrukciójával az, hogy a *ház* a *modellezett rendszer*nek, míg a *tervrajz* a *modell*nek felelhet meg, és az *elemek, melyek a tervrajzban használatosak* az UML -beli *modell elemeknek* megfelelői.

A következő szakaszok a Modell Menedzsment csomag absztrakt szintaxisát, jól képzett szabályait és szemantikáját írják le.

.2 Absztrakt szintaxis

A Modell Menedzsment csomag absztrakt szintaxisának grafikus szemléltetése a 23. ábrán látható.



23. ábra: Modell Menedzsment

A Modell Menedzsment csomag a következő metaosztályokat tartalmazza:

ElemHivatkozás (ElementReference)

Egy *elem hivatkozás* a csomag által hivatkozott modell elem láthatóságát és fedőnevét (alias -át) definiálja.

A metamodellben az ***ElemHivatkozás*** egy ***Csomag*** és egy ***ModellElem*** közötti kapcsolatot testesíti meg. Fedőnevet definiál a ***ModellElem***nek a ***Csomag***on belül; valamint meghatározza a ***ModellElem Csomag***ra vonatkoztatott láthatóságát.

Attributumok (Attributes)

fedőnév (alias) A fedőnév a hivatkozott ***ModellElem*** helyi nevét definiálja a ***Csomag***on belüli felhasználhatóság érdekében.

láthatóság (visibility) Minden hivatkozott ***ModellElem*** vagy publikus, vagy védett, vagy privát, összeköttetésben a hivatkozott ***Csomag***gal.

Asszociációk (Associations)

Nincs külön asszociáció.

Modell

A *modell* a modellezett rendszer egy absztrakciója, egy meghatározott nézőpontból és absztrakciós szinten specifikálja a modellezett rendszert. Egy modell abban az esetben tekinthető befejezettnek, ha teljes mértékben leírja az egész modellezett rendszert a választott absztrakciós szinten illetve nézőpontból.

A metamodellben a ***Modell*** a ***Csomag*** alosztálya. A ***ModellElem***ek egy tartalmazási hierarchiáját foglalja magába, melyek együttesen írják le a modellezett rendszert. Egy ***Modell*** tartalmazhat még olyan ***ModellElem***eket is, mint a ***Szereplő***k, melyek a rendszer környezetét reprezentálják a kölcsönhatásatásaival együtt, mint amilyenek a ***Függőség***ek, ***Általánosítás***ok és ***Megszorítás***ok.

Különböző ***Modell***ek definiálhatók ugyanahhoz a modellezett rendszerhez, így különböző nézőpontból határozhatjuk meg azt. Ilyen nézőpontok lehetnek pl.: logikai modell, tervezési modell, használati eset modell, stb. Minden ***Modell*** öntartalmazó a modellezett rendszer egy adott nézőpontjából és a választot absztrakciós szinten belül.

Attributumok (Attributes)

Nincs külön attributum.

Asszociációk (Associations)

Nincs külön asszociáció.

Csomag (Package)

A *csomag* nem más, mint modell elemek csoportosítása.

A metamodellben a ***Csomag ÁltalánosíthatóElem***. Egy ***Csomag*** olyan ***ModellElem***eket tartalmaz, mint ***Csomag***ok, ***Osztályozó***k és ***Asszociáció***k. Egy ***Csomag*** tartalmazhatja még a ***Csomag ModellElem***ei közötti ***Megszorítás***okat és ***Függőség***eket is.

Egy ***Csomag***nak lehetnek ***<<import>>*** függőségei más ***Csomag***okhoz, lehetővé téve ezzel, hogy az első ***Csomag***beli ***ModellElem***ek felhaszánálhassák a más ***Csomag***okon belüli ***ModellElem***eket. Egy ***Csomag***on belül rendelkezésre álló ***ModellElem***eket az aktuális ***Csomag*** a hivatkozott ***Csomag***okkal együttesen birtokolja, azaz más importált ***Csomag*** is birtokolja. Továbbá egy ***Csomag*** minden ***ModellElem***ének van a ***Csomag***ra vonatkoztatott *láthatósága*. E *láthatóság* megadja, hogy a ***ModellElem*** a ***Csomag***on kívülről is láthatók -e.

Attributumok (Attributes)

Nincs külön attributum.

Asszociációk (Associations)

hivatkozottElem (referencedElement) Egy ***Csomag*** más, importált ***Csomag*** ***ModellElem***eire hivatkozik.

Alrendszer (Subsystem)

Az *alrendszer* modell elemek csoportosítása, melyek közül néhány a más tartalmazott modell elem által felkínált viselkedés specifikációját képezi.

A metamodellben az ***Alrendszer*** a ***Csomag*** és az ***Osztályozó*** alosztálya, melynek ***Tulajdonság***ai mind ***Operáció***k. Az ***Alrendszer*** tartalma két részhalmazra oszlik: specifkációs elemek és realizációs elemek. Az előbbi az ***Alrendszer Operáció***ival együtt biztosítja az ***Alrendszer*** által tartalmazott viselkedés specifikációját, míg a ***ModellElem***ek az utóbbi halmazban közösen biztosítják a specifikáció realizációját. A specifikációs elemek ***HasználatiEset***ek az általuk felkínált ***Interfész***ekkel, ***Megszorítás***okkal és kapcsolatokkal együttesen. A realizációs elemek az ***Osztály***ok és az ***Alrendszer***ek a velük kapcsolatban álló ***Interfész***ekkel, ***Megszorítás***okkal és kapcsolatokkal együtt. A specifikációs és a realizációs elemek közötti kapcsolat ***Együttműködés***ek egy csoportjával van definiálva.

Attributumok (Attributes)

Példányosítható Megadja, hogy egy ***Alrendszer*** példányosítható -e. Ha igaz, akkor a modell elemek példányai egy alrendszeren belül egy implicit kompozíciót alkotnak egy implicit alrendszer példányhoz, függetlenül attól, hogy az ténylegesen implementált -e.

Asszociációk (Associations)

Nincs külön asszociáció.

.3 Jól képzett szabályok

A következő jól képzett szabályokat alkalmazzuk a Modell Menedzsment csomaghoz.

ElemHivatkozás (ElementReference)

Nincs külön jól képzett szabály.

Modell

Nincs külön jól képzett szabály.

Csomag (Package)

[1] Egy ***Csomag*** csak a következőket birtokolhatja, illetve utalhat rá: ***Csomag***ok, ***Alrendszer***ek, ***Osztályozó***k, ***Asszociáció***k, ***Általánosítás***ok, ***Függőség***ek, ***Megszorítás***ok, ***Együttműködés***ek, ***Üzenet***ek és ***Sztereotípus***ok.

```
self.contents->forAll ( c |
      c.oclIsKindOf(Package)         or
      c.oclIsKindOf(Subsystem)              or
      c.oclIsKindOf(Classifier)      or
      c.oclIsKindOf(Association)     or
      c.oclIsKindOf(Generalization)  or
      c.oclIsKindOf(Dependency)      or
      c.oclIsKindOf(Constraint)      or
      c.oclIsKindOf(Collaboration)   or
      c.oclIsKindOf(Message)         or
      c.oclIsKindOf(Stereotype) )
```

[2] A hivatkozott elemnek (kivéve az ***Asszociáció***t) nem lehet ugyanaz a neve vagy fedőneve, mint a ***Csomag*** vagy annak egy sztereotípusa által birtokolt elemé.

```
self.allReferencedElements->reject( re |
  re.oclIsKindOf(Association) )->forAll( re |
     (re.elementReference.alias <> '' implies
        not (self.allContents - self.allReferencedElements)->reject( ve |
           ve.oclIsKindOf (Association) )->exists ( ve |
              ve.name = re.elementReference.alias))
     and
     (re.elementReference.alias = '' implies
        not (self.allContents - self.allReferencedElements)->reject ( ve |
           ve.oclIsKindOf (Association) )->exists ( ve |
              ve.name = re.name) ) )
```

[3] A hivatkozott elemnek (kivéve az ***Asszociáció***t) nem lehet ugyanaz a neve vagy fedőneve.

```
self.allReferencedElements->reject( re |
  not re.oclIsKindOf (Association) )->forAll( r1, r2 |
     (r1.elementReference.alias <> '' and r2.elementReference.alias <> '' and
        r1.elementReference.alias = r2.elementReference.alias implies r1 = r2)
     and
     (r1.elementReference.alias = '' and r2.elementReference.alias = '' and
        r1.name = r2.name implies r1 = r2)
     and
     (r1.elementReference.alias <> '' and r2.elementReference.alias = '' implies
        r1.elementReference.alias <> r2.name))
```

[4] A hivatkozott elemnek (kivéve az ***Asszociáció***t) nem lehet ugyanaz a neve vagy fedőneve - kombinálva a kapcsolódó ***Osztályozó***k ugyanazon halmazával - mint a ***Csomag*** vagy annak egy sztereotípusa által birtokolt bármely ***Asszociáció***é.

```
self.allReferencedElements->select( re |
  re.oclIsKindOf(Association) )->forAll( re |
     (re.elementReference.alias <> '' implies
        not (self.allContents - self.allReferencedElements)->select( ve |
           ve.oclIsKindOf(Association) )->exists( ve : Association |
              ve.name = re.elementReference.alias
              and
              ve.connection->size = re.connection->size and
              Sequence {1..re.connection->size}->forAll( i |
                 re.connection->at(i).type = ve.connection->at(i).type ) ) )
     and
     (re.elementReference.alias = '' implies
        not (self.allContents - self.allReferencedElements)->select( ve |
           not ve.oclIsKindOf(Association) )->exists( ve : Association |
              ve.name = re.name
              and
              ve.connection->size = re.connection->size and
              Sequence {1..re.connection->size}->forAll( i |
                 re.connection->at(i).type = ve.connection->at(i).type ) ) ) )
```

[5] A hivatkozott elemnek (kivéve az ***Asszociáció***t) nem lehet ugyanaz a neve vagy fedőneve kombinálva a kapcsolódó ***Osztályozó***k ugyanazon halmazával.

```
self.allReferencedElements->select ( re |
  re.oclIsKindOf (Association) )->forAll ( r1, r2 : Association |
     (r1.connection->size = r2.connection->size and
     Sequence {1..r1.connection->size}->forAll ( i |
        r1.connection->at (i).type = r2.connection->at (i).type and
     r1.elementReference.alias <> '' and r2.elementReference.alias <> '' and
      r1.elementReference.alias = r2.elementReference.alias implies r1 = r2))
     and
     (r1.connection->size = r2.connection->size and
       Sequence {1..r1.connection->size}->forAll ( i |
          r1.connection->at (i).type = r2.connection->at (i).type and
        r1.elementReference.alias = '' and r2.elementReference.alias = '' and
         r1.name = r2.name implies r1 = r2))
     and
     (r1.connection->size = r2.connection->size and
     Sequence {1..r1.connection->size}->forAll ( i |
        r1.connection->at (i).type = r2.connection->at (i).type and
     r1.elementReference.alias <> '' and r2.elementReference.alias = '' implies
        r1.elementReference.alias <> r2.name)))
```

[6] Egy ***Csomag*** hivatkozott elemei tranzitíven publikus elemei az importált ***Csomag***oknak.

```
self.referencedElement = self.requirement->select (d |
  d.stereotype.name = 'import').supplier.oclAsType(Package).allVisibleElements
```

[7] Egy ***Csomag*** minden birtokolt ***Csomag***ját importálja.

```
self.requirement->select (s |
  s.stereotype.name = 'import').supplier->includesAll(
        self.ownedElement->select ( e | e.oclIsKindOf (Package) ) )
```

Kiegészítő Operációk

[1] A *tartalomjegyzék (contents)* művelet olyan halmazt eredményez, mely tartalmazza a ***Csomag*** által birtokolt vagy importált ***ModellElem***eket.

```
contents : Set(ModelElement)
contents = self.ownedElement->union(self.referencedElement)
```

[2] A *mindenHivatkozottElem (allReferencedElement)* művelet olyan halmazt eredményez, mely tartalmazza a ***Csomag*** vagy szupertípusai egyike által hivatkozott ***ModellElem***eket.

```
allReferencedElements : Set(ModelElement)
allReferencedElements = self.referencedElement->union(
  self.supertype.oclAsType(Package).allReferencedElements->select( re |
     re.elementReference.visibility = #public or re.elementReference.visibility
= #protected))
```

Alrendszer (Subsystem)

[1] Minden ***Operáció***hoz egy - valamely ***Alrendszer*** által felkínált - ***Interfész***ben magának az ***Alrendszer***nek, vagy legalább egy tartalmazott ***HasználatiEset***nek kell rendelkeznie egy illeszkedő ***Operáció***val.

```
self.specification.allOperations->forAll(interOp |
  self.allOperations->union(self.allSpecificationElements.allOperations)->exists
```

```
        ( op | op.hasSameSignature(interOp) ) )
```

[2] Egy ***Alrendszer Tulajdonság***ai csakis ***Operáció***k lehetnek.

```
self.feature->forAll(f | f.oclIsKindOf(Operation))
```

[3] Minden ***Operáció***t egy ***Együttműködés***nek kell realizálnia.

```
not self.isAbstract implies self.allOperations->forAll( op |
   self.allContents->select(c |
         c.oclIsKindOf(Collaboration) )->exists(c : Collaboration|
                                          c.representedOperation = op ) )
```

[4] Minden specifikációs elemet egy ***Együttműködés***nek kell realizálnia.

```
not self.isAbstract implies self.allSpecificationElements->forAll( s |
   self.allContents->select(c |
      c.oclIsKindOf(Collaboration) )->exists(c : Collaboration|
            c.representedClassifier = s ) )
```

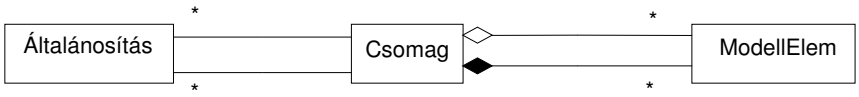
Kiegészítő Operációk

[1] Az *összesSpecifikációsElem* operáció olyan halmazt eredményez, mely az ***Alrendszer*** viselkedését specifikáló ***ModellElem***eket tartalmazza.

```
allSpecificationElements : Set(UseCase)
allSpecificationElements = self.allContents->select(c | c.oclIsKindOf(UseCase) )
```

.4 Szemantika

Csomag (Package)



A *csomag* konstrukció célja az, hogy egy általános csoportosító mechanizmust biztosítson. Egy csomag nem példányosítható, így tehát nem rendelkezik futásidejű szemantikával; valójában a szemantikája csak abból áll, hogy névterületet definiál a tartalmához. A *csomag* konstrukció bármilyen célú elem szervezéshez felhasználható; az, hogy olyan elemeket szervezzünk egy csoportba, melyek nem definiáltak az UML -en belül.

Egy csomag a modell elemek egy csoportját birtokolhatja. Az ugyanazon csomag által birtokolt elemeknek egyedi nevekkel kell rendelkezniük a csomagon belül, azonban különböző csomagokon belüli elemek már rendelkezhetnek ugyanazzal a névvel.

Ugyanazon csomag által tartalmazott modell elemek között is lehetnek kapcsolatok, de nem a priori egy csomagbeli elem és egy külső elem között. Más szóval a csomagon kívüli elemek alapértelmezés szerint nem állnak rendelkezésre a csomagon belüli elemek számára. Két eljárás létezik arra, hogy elérhetővé tegyük őket a csomag

belsejében: importáljuk az őket tartalmazó csomagokat, vagy általánosításokat definiálunk (ld. lentebb) ezen külső elemeket tartalmazó csomagokat. Az *import függőség* (egy ***Függőség*** az ***<<import>>*** sztereotípussal) egyik csomagtól a másikig azt fejezi ki, hogy az első csomag *hivatkozik* minden a második csomagbeli elegendő láthatósággal rendelkező elemre. A hivatkozott elemeket nem birtokolja a csomag, de felhasználhatók asszociációkban, általánosításokban, attributum típusokban és más kapcsolatokban. Egy csomag az általa tartalmazott elemek következő fajta *láthatóságait* definiálja: privát, védett, publikus. A *privát* -ként definiált elemek nem érhetők el az őket tartalmazó csomag külvilága által, a *védett* elemek csak az őket tartalmazó csomag általánosításaiként előállt csomagok által érhetőek el, míg a *publikus* elemek még az importáló csomagok által is elérhetőek. Megjegyzendő azonban, hogy a láthatósági mechanizmus nem korlátozza egy elem láthatóságát ugyanazon csomag egyenrangú elemei szempontjából.

Amikor egy csomag hivatkozik egy elemre, akkor ez az elem kiterjeszti a csomag névterületét. Ez lehetővé teszi azt, hogy a hivatkozott elemnek *fedőnevet (aliast)* adunk, így nem fog ütközni más elemek nevével a névterületen belül. A fedőnév az elem neve lesz a névterületben; az elem nem fog felbukkanni az eredeti és a fedőneve alatt egyszerre. Ha egy elemhez nincs megadva fedőnév, akkor a teljes útvonal-nevével kell azonosítani, azaz az őt körülvevő csomagok neveinek összevonásával, a legfelső szintű csomagnál kezdve. Továbbá egy elemnek ugyanolyan, vagy korlátozottabb láthatósággal kell rendelkeznie egy csomagban, mely hivatkozik rá, mint amilyen láthatósággal az őt birtokló csomag rendelkezik, azaz egy elem, mely az egyik csomagban publikus, védett vagy privát lehet egy rá hivatkozó csomag számára.

Egy másik csomagot importáló csomag az importált csomag által definiált névterület összes publikus tartalmára hivatkozik, beleértve az importált csomag által importált csomagok elemeit is. Ebből az következik, hogy a csomagok importja tranzitív, ezt szemlélteti a következő példa:

Tegyük fel, hogy **A** csomag importálja **B** csomagot, mely ezután importálja **C** csomagot. Ekkor **C** publikus elemei publikusak **B** -ben és rendelkezésre állnak **A** számára is.

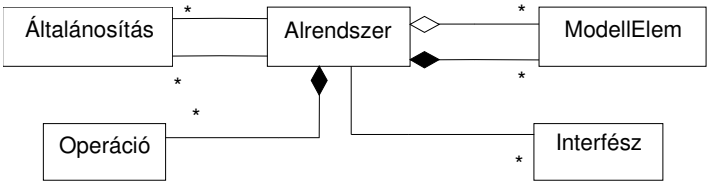
A csomagokat automatikusan importálják az őket tartalmazó csomagok. Az importálás rekurzivitása következtében még a csomagok több szintjén belül tartalmazott elemek is rendelkezésre állnak a tartalmazott elemek láthatóságának megfelelően. Egy olyan elem láthatósága, mely a csomagok több szintjén belül is tárolódik, a legkorlátozottabb az összes tároló csomag közül.

Egy csomagnak lehetnek *általánosításai* más csomagokhoz. Ez azt jelenti, hogy a publikus és a védett elemek, melyeket birtokol, vagy melyekre hivatkozik egy csomag, rendelkezésre állnak az ő örököseik számára is, és ugyanolyan módon használhatók, mint maguk az örökösök által hivatkozott bármely elem. Azok az elemek, melyek rendelkezésre állnak egy általánosítás által használt más csomag számára is, valós nevükön, nem pedig fedőnevükön jelennek meg. Továbbá, ugyanaz a láthatóságuk az örökösben, mint az őket birtokló csomagban.

Egy csomag egy *keretrendszer (framework)* defininálására is felhasználható, mely mintákból áll. Jó példát képeznek az együttműködések, ahol az alapelemek (némelyike)

a minták paramétereit képezik. Ettől eltekintve egy keretrendszer csomag közönséges csomagként van leírva.

Alrendszer (Subsystem)



Az *alrendszer* konstrukció célja biztosítani egy csoportosító mechanizmust azzal a lehetőséggel, hogy tartalmának viselkedését specifikálni lehet. Egy alrendszer lehet példányosítható és lehet nem példányosítható. Egy nem példányosítható alrendszer csupán névterületet definiál a tartalma számára. Egy alrendszer tartalma ugyanazzal a szemantikával rendelkezik egy csomag, így az birtokolt elemekből és hivatkozott elemekből áll egyedi nevekkel vagy fedőnevekkel az alrendszeren belül.

Egy alrendszer tartalma két részhalmazra osztható: a *specifikációs* elemekére és a *realizációs* elemekére. A specifikációs elemeket arra használjuk, hogy a realizációs elemek által felkínált viselkedés absztrakt specifikációját adjuk meg általuk.

Egy alrendszer specifikációja tartalmának specifikációs részhalmazából és az alrendszer *tulajdonságaiból* (operációk) áll. Specifikálja a realizációs részhalmazbeli osztályozók példányai által együttesen végrehajtott viselkedést anélkül, hogy bármit is felfedne e részhalmaz tartalmából. A specifikáció a használati esetek és/vagy operációk szempontjából, ahol a használati esetek az alrendszer által (pontosabban tartalmának példányai által) végrehajtott teljes szekvenciák specifikációjára használatosak, kölcsönhatásban a környezettel, míg az operációk csak bizonyos részeket specifikálnak. Továbbá egy alrendszer specifikációs része magába foglal még megszorításokat, használati esetek közötti kapcsolatokat, stb.

Egy alrendszernek nincsen saját viselkedése. Az alrendszer specifikációjában definiált összes viselkedést tartalmának realizációs részhalmaza együttesen kínálja fel. Általában, mivel ezek osztályozók, az alrendszerek bárhol előfordulhatnak, ahol osztályozó kívánatos. Ennek az általános interpretációja, hogy mivel az alrendszer önmaga nem példányosítható, illetve nincs saját viselkedése, az alrendszerre vonatkozó követelményeket abban a kontextusban, ahol előfordul, a tartalma teljesíti. Ugyanez igaz az asszociációkra is; azaz bármely egy alrendszerhez csatlakozó asszociáció ténylegesen csatlakozik a tartalmát képező osztályozók egyikéhez.

Egy alrendszer specifikációs része és realizációs része közötti megfeleltetést *együttműködések* halmaza specifikálja, legalább egy az alrendszer minden operációjához és minden tartalmazott használati esethez. Minden együttműködés specifikálja, hogy a realizációs elemek példányai miként kooperáljanak a használati eset vagy operáció által specifikált viselkedés együttes végrehajtásához, azaz hogyan kell a magasabb szintű absztrakciót transzformálni alacsnyabb szintű absztrakcióvá. Egy használati eset (a magasabb absztrakciós szinten) példánya által fogadott üzenet példány megfelel azon együttműködésbeli osztályozó szerephez igazodó példánynak, mely együttműködés fogadja az üzenet példányt (alsó absztrakciós szint). E példány

kommunikál más példányokkal, melyek idomulnak más osztályozó szerepekhez az együttműködésben, és együtt hajtják végre azt a viselkedést, melyet a használati eset határozott meg. Az összes üzenet példányt, melyet a használati esetek példányai fogadtak, illetve küldtek, fogadják, ill. küldik a megfelelő példányok is, de az alsó absztrakciós szinten. Hasonlóan, az alrendszer egy operációjának alkalmazása ténylegesen azt jelenti, hogy egy üzenet példány kerül elküldésre a tartalmazott példányhoz, mely ennek hatására végrehajt egy metódust.

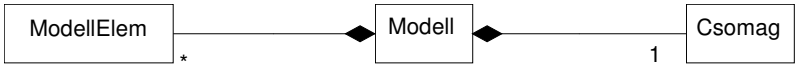
Az alrendszerek *importálása* ugyanolyan módon történik, mint a csomagoknál, felhasználva a *láthatóság* tulajdonságot annak definiálására, hogy az alrendszer elemei *publikusak, védettek* vagy *privátak*.

Egy alrendszer rendelkezhet *általánosításokkal* más rendszerekhez. Ez azt jelenti, hogy a publikus és védett elemek egy alrendszer tartalmában rendelkezésre állnak az örökösei számára is.

Az alrendszerek *interfészeket* is felkínálhatnak. Ez azt jelenti, hogy minden az interfészben definiált, az interfészt felkínáló alrendszerhez kell léteznie egy illeszkedő operációnak, mint magának az alrendszernek, vagy egy használati esetnek egy tulajdonsága. Nem szükséges, hogy az interfész és az alrendszer közötti kapcsolat egy - egy kapcsolat legyen; egy alrendszer több interfészt is megvalósíthat, és egy interfészt megvalósíthat egynél több alrendszer is.

Egy alrendszer felhasználható *keretrendszer* definiálásához is. Egy keretrendszert definiáló alrendszer specifikációja lehet paraméterezett is.

Modell



A *modell* célja a modellezett rendszer leírása egy bizonyos absztrakciós szinten és egy bizonyos nézőpontból, mint amilyen a modellezett rendszer logikai, vagy viselkedési nézőpontja.

Egy modell teljesen leírja a modellezett rendszert abban az értelemben, hogy a választott absztrakciós szinten és nézőpontból lefedi a teljes modellezett rendszert. A modellt egy tartalmazási hierarchia alkotja, ahol a legfelső szintű csomag reprezentálja a modellezett rendszer határát. A környezet modellezhető a szereplőkkel és azok interfészeivel. Ez utóbbi modell elemek és a modellezett rendszert reprezentáló modell elemek összekapcsolhatók egymással. Ilyen kapcsolatokat birtokolhat a modell, vagy a legfelső szintű csomag. Egy modell tartalma a birtokolt modell elemek tranzitív egybezárása. Ilyen modell elemek a csomagok, osztályozók és kapcsolatok.

A különböző modellekbeli modell elemek közötti kapcsolat nincs kihatással a modell elemek jelentésére az őket tartalmazó modellen belül a modellek öntartalmazása miatt. Megjegyzendő azonban, hogy még ha a modellközi kapcsolatok nem is fejeznek ki szemantikát a modellekre vonatkozólag, rendelkeznek szemantikával az olvasóra, vagy a származtatott modell elemekre, mint az egész fejlesztési folyamat részeire nézve.

Egy modell lehet más modell specializációja. Ez magába foglalja, hogy az összes elem az ősben rendelkezésre áll a specializált modellben is, ugyanazon név alatt, mint az ősben.

.5 Szabványos elemek

Az előre definiált sztereotípusok, megszorítások és csatolt értékek a Modell Menedzsment csomaghoz a következő táblázatban láthatók, illetve megtalálhatók a *Szabványos Elemek* függelékben.

Modell Elem	Stereotípusok	Megszorítások	Csatolt Értékek
Csomag	«facade» (arculat) «framework» (keretrend.) «stub» «system» (rendszer)		

Modell Menedzsment - Szabványos Elemek

.6 Megjegyzések

Annak következtében, hogy e modell az UML egy logikai modellje, a különféle eszközök közötti disztribúció vagy megosztás leírását nem tartalmazza.

Egy elem *láthatósága* egy importáló csomagban/alrendszerben lehet korlátozóbb, mint az őt birtolkó névterületen belül. Ez igen hasznos például akkor, amikor egy névterület publikussá teszi tartalmát a környező névterület számára, de ezen elemek nem elérhetők a környező névterületen kívülről.

Az UML -en belül három különböző módszer létezik olyan elemek csoportjának modellezésére, melyeket más elem tartalmaz: *csomag, alrendszer* vagy *osztály* használata. A következőkben néhány szabályt adunk ezek használatához:

- Csomagot akkor használunk, amikor semmi másra nincs szükség, mint a szükséges elemek egyszerű csoportba szervezése.

- Az alrendszerek megfelelő csoportosítást biztosítanak a “felülről - lefelé” módszerű fejlesztéshez, mivel a tartalmukra vonatkozó követelmények kifejezhetők e viselkedés realizációjának definiálása előtt.

- Osztályokat olyankor használunk, amikor magának a konténernek példányosíthatónak kell lennie, így lehetséges a kompozit objektumok definiálása.

6. FÜGGELÉKEK

A következő függelékek a szabványos elemek összefoglalását, valamint egy szójegyzéket biztosítanak.

Tartalom

1 “A” FÜGGELÉK: Szabványos Elemek

Ez a függelék leírja az UML előre definiált szabványos elemeit. A szabványos elemek kategóriákba vannak szervezve (sztereotípusok, csatolt értékek és megszorítások) és ABC szerint rendezettek.

.1 Sztereotípusok

Az UML előre definiált sztereotípusait a következő táblázat tartalmazza. E sztereotípusok közül bármelyik, amelyik egy meghatározott osztályhoz alkalmazható a metamodellben, a szóbanforgó osztály bármelyik alosztályához is alkalmazható.

Név	Mihez alkalmazzuk?	Leírás
«becomes»	Függőség	A«becomes» egy sztereotípizált függőség, melynek forrása és célja ugyanazt a példányt reprezentálja különböző időpontokban, de potenciálisan különböző értékekkel, állapot példányokkal és szerepekkel. Egy«becomes» függőség **A** -tól **B -**ig azt jelenti, hogy az **A** példány **B** -vé alakul át, esetleg új értékekkel, állapot példánnyal és szerepekkel, különböző időbeli és térbeli pozícióval.
«call»	Függőség	A «call» egy sztereotípizált függőség, melynek forrása és célja egy - egy operáció. Egy «call» függőség meghatározza, hogy a forrás meghívja a cél operációt. Egy «call» hozzákapcsolhat egy forrás operációt bármely - a hatókörén belüli - cél operációhoz, beleértve, de nem korlátozva az őt körülvevő, illetve más látható osztályozó operációit.
«copy»	Függőség	A «copy» egy sztereotípizált függőség, melynek forrása és célja különböző példányok, de ugyanazokkal az értékekkel, állapot példányokkal és szerepekkel (viszont eltérő azonosítóval). Egy «copy» függőség **A** -tól **B** -ig azt fejezi ki, hogy **B** egy pontos másolata **A**-nak. A jövőbeli változások **A** -ban nem szükségszerűen hatnak ki **B** -re.

«create»	ViselkedésiTulajdonság	A «create» egy sztereotípizált viselkedési tulajdonság, azt jelenti, hogy a kijelölt tulajdonság létrehozza az osztályozó egy példányát, melyhez e tulajdonság csatlakozni fog.
	Esemény	A «create» egy sztereotípizált esemény, azt jelenti, hogy a példány, ami körülveszi azt az állapot gépet, melyhez az esemény típust alkalmazzuk, létrejött. A «create» sztereotípust csak az említett állapot gép legfelső szintjén egy kezdeti átmenethez használhatjuk. Ez valójában csak a trigger egy fajtája, melyet egy kezdeti átmenethez használhatunk.
«destroy»	ViselkedésiTulajdonság	A «destroy» egy sztereotípizált viselkedési tulajdonság, azt jelenti, hogy a kijelölt tulajdonság leromolja az osztályozó egy példányát, melyhez a tulajdonság kapcsolódott.
	Esemény	A «destroy» egy sztereotípizált esemény azt jelenti, hogy a példány, ami körülveszi azt az állapot gépet, melyhez az esemény típust alkalmazzuk, megsemmisült (lerombolódott).
«deletion»	Finomítás	A «deletion» egy sztereotípizált finomítás, mely nem rendelkezik kliensekkel és al-finomításokkal.
«derived»	Függőség	A «derived» egy sztereotípizált függőség, melynek forrása és célja egyaránt elemek, melynek típusa általában, de nem szükségszerűen ugyanaz. A «derived» függőség meghatározza, hogy a forrás a célból származik, ez pedig azt fejezi ki, hogy a forrás nem nyilvánvaló, hanem implicit módon származik a célból.
«document»	Komponens	A «document» egy sztereotípizált komponens, mely egy dokumentumot reprezentál.
«enumeration»	AdatTípus	Az «enumeration» egy sztereotípizált adat típus, melynek részletei egy olyan domént határoznak meg mely azonosítókból áll. Ezek az azonosítók az adat típus egy példányának lehetséges értékei.
«executable»	Komponens	Az «executable» egy sztereotípizált komponens, azt a programot jelenti, mely egy Csomóponton futhat.
«extends»	Általánosítás	Az «extends» egy sztereotípizált általánosítás a haszálati esetek között. Meghatározza, hogy a kiterjesztési használati eset tartalma hozzáadható egy

		vele kapcsolatban álló haszálati esethez. Nem csak a tartalom hozzáadásának helyét (*kiterjesztésiPont*) határozza meg, hanem a hozzáadás *feltételét* is. E feltétel egy logikai kifejezés. Amikor a használati esettel összeköttetésben álló példány elérte a kiterjesztési pontot és a feltétel teljesült, a példány annak a szekvenciának megfelelően folytatja tevékenységét, mely az eredeti szekvencia kiterjesztő szekvenciával - ennél a pontnál történő - bővítésének eredménye. A kiterjesztő használati eset részeinek rendezetteknek kell lenniük abban az esetben, ha a részek különböző helyeken lesznek beillesztve.
«facade»	Csomag	A«facade» egy sztereotípizált csomag. Nem tartalmaz semmit, de más csomagok által birtokolt modell elemekre hivatkozik. Arra használjuk, hogy a csomg tartalma egy részéhez publikus nézőpontot biztosítsunk. Egy «facade» nem tartalmaz saját modell elemet.
«file»	Komponens	A «file» egy sztereotípizált komponens, forráskódot, vagy adatot tartalmazó dokumentumot reprezentál
«framework»	Csomag	A «framework» (keretrendszer) egy sztereotípizált csomag, mely főleg mintákat tartalmaz.
«friend»	Függőség	A«friend» egy sztereotípizált használati függőség, melynek forrása egy modell elem, mint amilyen egy operáció, osztály, vagy csomag, és melynek célja egy különböző csomag modell elem, mint amilyen egy osztály, vagy csomag. Egy «friend» kapcsolat biztosítja, hogy a forrás - a deklarált láthatóságtól függetlenül - elérje a célt. Kiterjeszti a forrás láthatóságát, így a cél láthatja a forrást.
«import»	Függőség	Az «import» egy sztereotípizált függőség két csomag között, azt jelenti, hogy a cél csomag publikus tartalma hozzáadódik a forrás névterületéhez.
«implementationClass»	Osztály	Az«implementationClass» egy sztereotípizált osztály, mely nem egy típus, és mely valamely programnyelvbeli osztály implementációját reprezentálja. Egy példánynak lehet nulla, vagy implementációs osztálya. Ez ellentettje a sima általános osztályoknak, melyekben egy példánynak statikusan több osztálya

		lehet egy időben, és szerezhet, ill. elveszíthet osztályokat az idő során; ezen kívül pedig egy objektum (a példány altípusa) dinamikusan rendelkezhet több osztállyal is.
«inherits»	Általánosítás	Az «inherits» egy sztereotípizált általánosítás, azt fejezi ki, hogy az altípus példányai nem helyettesíthetők a szupertípus példányával.
«instance»	Függőség	Az «instance» egy sztereotípizált függőség, melynek forrása egy példány és célja egy osztályozó. Egy **I** -től **C** -ig példány függőség azt fejezi ki, hogy **I** egy példánya **C** -nek.
«invariant»	Megszorítás	Az «invariant» egy sztereotípizált megszorítás, melyet osztályozókhoz vagy kapcsolatokhoz kell illeszteni. Azt jelenti, hogy a megszorítás feltételeit meg kell tartani az osztályozókhoz vagy kapcsolataikhoz és példányaikhoz.
«library»	Komponens	A «library» egy statikus vagy dinamikus (mint amilyen a Windowsban a DLL) könyvtárat reprezentáló sztereotípizált komponens.
«metaclass»	Függőség	A «metaclass» egy sztereotípizált függőség, melynek forrásai és céljai egyaránt osztályozók. Azt fejezi ki, hogy a cél a forrás metaosztálya.
	Osztályozó	A «metaclass» egy sztereotípizált osztályozó, azt fejezi ki, hogy az osztály néhány más osztály metaosztálya.
«postcondition»	Megszorítás	A «postcondition» egy sztereotípizált megszorítás, melyet egy operációhoz kell kapcsolni. Azt jelenti, hogy a megszorítás feltételeit meg kell tartani az operáció meghívása után.
«powertype»	Osztályozó	A «powertype» egy sztereotípizált osztályozó. Azt fejezi ki, hogy az osztályozó egy metatípus, melynek példányai mástípus altípusai.
	Függőség	A «powertype» egy sztereotípizált függőség, melynek forrását az általánosítások halmaza képezi, célja pedig egy olyan osztályozó, mely meghatározza, hogy a cél a forrás hatványtípusa.
«precondition»	Megszorítás	A «precondition» egy sztereotípizált megszorítás. Azt jelenti, hogy a megszorítás feltételeit meg kell tartani az operáció meghívásához.
«private»	Általánosítás	A «private» a privát öröklődés

		sztereotípizált általánosítása. Eljrejti az osztály örökölt tulajdonságait és ezért nem helyettesíthetővé teszi az ősei deklarációi számára.
«process»	Osztályozó	A «process» egy sztereotípizált osztályozó, mely ezen kívül még egy aktív osztály is. Egy "nehézsúlyú" vezérfolyamot reprezentál.
«requirement»	Megjegyzés	A «requirement» egy sztereotípizált megjegyzés, mely egy felelősséget vagy kötelezettséget ad meg.
«send»	Függőség	A «send» egy sztereotípizált függőség, melynek forrása egy operáció és célja egy jel. Meghatározza, hogy a forrás a cél jelet küldi.
«stereotype»	Osztályozó	A «stereotype» egy sztereotípizált osztályozó. Azt jelenti, hogy az osztályozó szterotípusként szolgál. Ez a sztereotípus lehetővő teszi a modellezők számára, hogy sztereotípus hierarchiákat modellezzenek.
«stub»	Csomag	A «stub» egy sztereotípizált csomag. Nem teljes mértékbe átvitt csomagot reprezentál. A «stub» kifejezetten a csomag publikus részeit biztosítja, de semmi többet.
«subclass»	Általánosítás	A «subclass» egy sztereotípizált általánosítás. Azt fejezi ki, hogy az altípus példányai nem helyettesíthetők a szupertípus példányaival.
«subtraction»	Finomítás	A «subtraction» egy sztereotípizált finomítás, mely nem rendelkezik kliensekkel és al-finomításokkal.
«subtype»	Általánosítás	A «subtype» egy sztereotítpizált általánosítás mely nem kínál fel különböző tulajdonságot vagy viselkedést, mint az alap általánosítás. E sztereotípusok az alosztály ellentettjeiként léteznek.
«system»	Csomag	A «system» egy sztereotípizált csomag, mely ugyanazon modellezett rendszer modelleinek kollekcióját reprezentálja. A «system» által tartalmazott modellek a modellezett rendszert különböző nézőpontokból írják le, e nézőpontoknak nem szükséges különálló halmazokat alkotniuk. A «system» éppenezért a modellezett rendszer átfogó specifikációját állítja össze - ez a legfelső szintű konstrukció a specifikációban. Egy «system» a különböző modellekbeli modell elemek között fennálló kapcsolatokat és megszorításokat is tartalmazza. E modell

		elemek nem adnak szemantikus információt a csatlakoztatott modell elemekhez, mivel minden modell a modellezett rendszer egy teljes nézőpontját ábrázolja. Így e modell elemek nem a modellezett rendszerre, hanem inkább magára a modellre vonatkozólag, fejeznek ki információt, azaz a követelmények nyomon követésére használhatók. Egy modellezett rendszert az *alárendelt* modellezett rendszerek halmaza is realizálhat. Minden ilyen *alárendelt* modellezett rendszert a saját, különálló «system» -ben összegyűjtött modellek halmaza ír le. Egy «system» csakis egy másik «system» -ben tárolódhat.
«table»	Komponens	A «table» egy adatbázis táblát reprezentáló sztereotípizált komponens.
«thread»	Osztályozó	A «thread» egy sztereotípizált osztályozó, mely ezen kívül még egy aktív osztály is. Egy "könnyűsúlyú" vezérfolyamot reprezentál.
«topLevelPackage»	Csomag	A «topLevelPackage» egy sztereotípizált csomag. Kijelöli a legfelső szintű csomagot egy modellben és reprezentálja a modell összes nem környezeti részét. Egy «topLevelPackage» egy modell tárolási hierarchiájának csúcsán helyezkedik el.
«type»	Osztály	A «type» az Osztály sztereotípusa. Azt jelenti, hogy az osztályt a (objektum) példányok egy doménjének specifikálására használjuk az objektumokhoz alkalmazható operációkkal együtt. Egy «type» semmilyen metódust nem tartalmazhat, attributumai és asszociációi lehetnek.
«useCaseModel»	Modell	A «useCaseModel» egy olyan modell, mely egy rendszer funkcionális követelményeit írja le a használati esetek és azok szereplőikkel való kölcsönhatásainak szempontjából. Fontos, hogy a «useCaseModel» csak használati eseteket és szereplőket, valamint a kapcsolataikat tárolhatja.
«uses»	Általánosítás	A «uses» egy sztereotípizált általánosítás a használati esetek között. Meghatározza, hogy a kapcsolatban álló használati eset tartalma más használati eset leírásában bent van -e. Ez kifejezetten a megosztott viselkedés kijelölésére haszálatos. A felhasznált használati esetek részeinek rendezetteknek kell lenniük, ha a részek

		felhasználása különböző helyeken történik. A «uses» csak használati esetek között definiálható.
«utility»	Osztályozó	A «utility» egy sztereotípizált osztályozó. Egy olyan osztályozót reprezentál, melynek nincsnek példányai, hanem inkább a nem tulajdonos attributumok és operációk névvel ellátott kollekcióját jelöli ki.

.2 Csatolt értékek

Az UML előre definiált csatolt értékeit a következő táblázat tartalmazza. E csatolt értékek közül bármelyik, amelyik egy meghatározott osztályhoz alkalmazható a metamodellben, a szóbanforgó osztály bármelyik alosztályához is alkalmazható.

Név	**Mihez alkalmazzuk?**	**Leírás**
documentation	Eleml	A documentation (dokumentáció) annak az elemnek a megjegyzése, leírása vagy magyarázata, amelyhez hozzákapcsolták.
location	Osztályozó	A location (elhelyezés) azt jelenti, hogy az osztályozó egy része a megadott komponensnek.
	Komponens	A location (elhelyezés) azt jelenti, hogy a komponens a megadott csomóponton tartózkodik.
persistence	Attributum	A persistence (megmaradás) azt jelenti, hogy az attributum állapotának végrehajtása átmenetinek (az állapota a példány lerombolásakor megszűnik) vagy fennmaradónak (az állapota a példány lerombolásakor nem szűnik meg) jelöli ki.
	Osztályozó	A persistence (megmaradás) azt jelenti, hogy az attributum állapotának végrehajtása átmenetinek (az állapota a példány lerombolásakor megszűnik) vagy fennmaradónak (az állapota a példány lerombolásakor nem szűnik meg) jelöli ki.
	Példány	A persistence (megmaradás) azt jelenti, hogy az attributum állapotának végrehajtása átmenetinek (az állapota a példány lerombolásakor megszűnik) vagy fennmaradónak (az állapota a példány lerombolásakor nem szűnik meg) jelöli ki.
responsibility	Osztályozó	A responsibility (felelősség) az osztályozó

		egy kötelezettsége.
semantics	Osztályozó	A semantics (szemantika, jelentés) az osztályozó jelentésének specifikációja.
	Operáció	A semantics (szemantika, jelentés) az osztályozó jelentésének specifikációja.

.3 Megszorítások

Az UML előre definiált megszorításait a következő táblázat tartalmazza.

Név	**Mihez alkalmazzuk?**	**Leírás**
association	KapocsVégpont	Az association egy a KapocsVégpontokhoz alkalmazott megszorítás. Meghatározza, hogy a megfelelő példányok láthatók az asszociáción keresztül.
broadcast	Kérelem	A broadcast (szétküldés, körbeküldés) megszorítást egy kérelem több példányhoz való elküldésére használunk. Meghatározza, hogy egyidejűleg, meghatározatlan sorrendben kell eljuttatni minden cél példányhoz.
complete	Általánosítás	A complete (befejezés) megszorítást általánosítások csoportjához alkalmazzuk. Meghatározza, hogy minden altípus specifikált és további altípusok nem engedélyezettek.
disjoint	Általánosítás	A disjoint (különálló) megszorítást általánosítások csoportjához alkalmazzuk. Meghatározza, hogy a példánynak nem lehet több, mint egy altípusa azon megadott altípusok közül, melyek a példány típusát képezik. Ez az általánosítás alapértelmezett szemantikája.
global	KapocsVégpont	A global (globális) egy olyan megszorítás, melyet a KapocsVégpontokhoz alkalmazunk. Meghatározza, hogy a megfelelő példány látható, mert egy globális hatáskörben van a kapocsra vonatkozólag.
implicit	Asszociáció	Az implicit egy olyan megszorítás, melyet egy asszociációhoz alkalmazunk. Meghatározza, hogy az asszociáció nem nyílt, hanem inkább csak fogalmi.
incomplete	Általánosítás	Az incomplete (befejezetlen) egy olyan megszorítás, melyet általánosítások csoportjához alkalmazunk. Meghatározza,

		hogy nem minden altípus specifikált és további altípusok engedélyezettek.Ez az általánosítás alapértelmezett szemantikája.
local	KapocsVégpont	A local (lokális) egy olyan megszorítás, melyet a KapocsVégpontokhoz alkalmazunk. Meghatározza, hogy a megfelelő példány látható, mert egy lokális hatáskörben van a kapocsra vonatkozólag.
or	Asszociáció	Az or (vagy) egy olyan megszorítás, melyet asszociációk halmazához alkalmazunk. Meghatározza, hogy e halmazon felül csak egy mutatkozik minden kapcsolódó példányhoz. Az or egy exkluzív (kizáró, nem megengedő) megszorítás.
overlapping	Általánosítás	Az overlapping (átfedés) egy olyan megszorítás, melyet általánosítások csoportjához alkalmazunk. Meghatározza, hogy a példányoknak lehet -e egynél több altípusa a megadottak - mint a példány lehetséges típusai - közül.
parameter	KapocsVégpont	A parameter (paraméter) egy olyan megszorítás, melyet a KapocsVégpontokhoz alkalmazunk. Meghatározza, hogy a megfelelő példány látható, mert az a parameter összeköttetésben álla kapoccsal.
self	KapocsVégpont	A self (önmaga) egy olyan megszorítás, melyet a KapocsVégpontokhoz alkalmazunk. Meghatározza, hogy a megfelelő példány látható, mert az egy kérelem diszpécsere (közvetítője).
vote	Kérelem	A vote (szavazat) egy olyan megszorítás, melyet egy kérelemhez alkalmazunk. Meghatározza azt a visszatérési értéket, melyet a több példányból érkező összes visszatérési érték többségi “szavazata” szelektál.

2 "B" FÜGGELÉK: Szójegyzék

MAGYAR	ENGLISH	LEÍRÁS
absztrakció	abstraction	Leegyszerűsítés. A vizsgált objektum - adott szempontból - lényegi tulajdonságainak kiválasztása, a lényegtelenek elhagyása. Az egyednek e lényegi tulajdonságai megkülönböztetik őt minden más egyedtől. Egy absztrakció egy - a szemlélő nézőpontjára vonatkozó - határt definiál.
absztrakt metódus	abstract method	Olyan üres virtuális metódus, mely csak örökítési célt szolgál, megteremtve ezzel a felülírás lehetőségét.
absztrakt osztály	abstract class	Olyan - absztrakt metódust tartalmazó - osztály, mely csak örökítési célt szolgál, példányt soha nem hoznak belőle létre (pl. alap osztály) Ellentéte: *konkrét osztály.*
adatszótár	data dictionary	Az analízis, tervezés során feltárt fogalmak magyarázó szótára.
adattípus	datatype	Olyan típus, mely értékeinek nincs azonosítója. Az adattípusok magukba foglalnak primitív beépített típusokat (mint pl. számokat és karakterláncokat), csakúgy mint felsorolt típusokat (amilyen pl. a logikai).
aggregáció	aggregation	Ld.: *tartalmazási kapcsolat*
aggregát [osztály]	aggregate [class]	Egy olyan osztály, mely az "egész" -et reprezentálja egy aggregátum (egész-rész) kapcsolatban. Ld.: *aggregátum*.
aggregát, összetett objektum	aggregate / composite / compound object	Tartalmazási kapcsolatban a tartalmazó, egész objektum.
aktív objektum	active object	Az aktív osztály egy példánya. Egy objektum, mely birtokol egy vezérfonalat, és képes inicializálni egy vezérlő tevékenységet. Ld. még: *aktív osztály, vezérfonál.*
aktív osztály	active class	Egy osztály, melynek példányai aktív objektumok. Ld.: *akív objektum*
aktiválás	activation	Egy hatás végrahajtása.
aktivitás diagram, hatásdiagram	activity diagram	Az állapotdiagram speciális esete, melyben mindegyik, vagy a legtöbb állapot hatásállapot és melyben mindegyik, vagy a legtöbb átmenet kiváltása (triggerelése) a forrásállapotbeli hatások befejezésével

		történik. Véges állapotú automata, egy operáció implementációját adja meg a meghívott operációk kifejtésével. Ellentéte: *állapot diagram.*
aktor, szereplő [osztály]	actor [class]	Szerepek összefüggő halmaza, melyeket a használati esetek használói játszanak amikor kölcsönhatást gyakorolnak ezekkel az esetekkel. Egy szereplőnek minden egyes alkalmazási esetre van egy szerepe, amellyel kommunikál. A rendszerre hatást gyakorló valós világbeli szereplő. Mindig ő gyakorol hatást egy másik objektumra, sosem fordítva. Közvetlen kapcsolatban áll a használati esetekkel.
aktuális paraméter	actual parameter	A kívülről elérhető metódus, eljárás, v. függvény hívásakor átadott paraméter. Szinonímája: *argumentum.*
alap osztály	base class	Az osztályhierarchia csúcsán lévő osztály(ok). Azt tartalmazza, ami minden osztályba szükséges.
alkalmazás	application	Komplett program, futtatható komponens. Egyedi elemekkel kiegészített keretrendszer.
állapot	state	Feltétel, vagy szituáció egy objektum élete folyamán, melyben kielégít bizonyos feltételt, végrehajt bizonyos tevékenységet, vagy vár valamilyen esemény bekövetkeztére. Ezt az állapotot az objektum adatainak pillanatnyi értékével fejezhetjük ki. Az objektum mindig emlékszik legutolsó állapotára.
állapot diagram	state diagram	Megadja, hogy egy adott objektum, adott esemény hatására milyen állapotból milyenbe kerül.
állapot gépezet	state machine	Viselkedés, mely az állapotok olyan szekvenciáját specifikálja, melyeken egy objektum vagy egy kölcsönhatás élettartama során átmegy. Beletartoznak az eseményekre adandó válaszok, valamint a hatásmechanizmusok.
állapotgrafikon diagram	statechart diagram	Állapotgépezetet ábrázoló diagram. Ld.: *állapotgépezet.*
alosztály	subclass	Más osztály, ill. szuperosztály specializációja egy általánosítási kapcsolatrendszeren belül. Az öröklődési ágon az aktuális osztály alatt levő osztály. Ld.: *általánosítás, utód osztály.* Ellentéte: *szuperosztály.*
alrendszer	subsystem	Az alrendszer modell elemek egy csoportosítása, mely a más tartalmazott modell elemek által felkínált viselkedés egy specifikációját képezi. Ld.: *csomag.* Ellentéte: *rendszer.*
általánosítás	generalization	Osztályozási kapcsolat egy általánosabb és egy speciálisabb elem között. A speciálisabb elem felülről kompatibilis az általánosabb elemmel, és további információt tartalmaz. Egy speciálisabb elem példánya ott használható, ahol az általánosabb elem is

		engedélyezett. Az elemekben azt keressük, hogy mi a közös bennük, és egy ilyen osztályt hozunk felettük létre. Ld.: *öröklődés.*
általánosítható elem	generalizable element	Modell elem, mely részt vehet egy általánosítási kapcsolatban. Ld.: *általánosítás.*
altípus	subtype	Egy másik típus, a szupertípus specializációja egy általánosítási kapcsolatrendszerben. Ld.: *általánosítás.* Ellentéte: *szupertípus.*
argumentum	argument	Egy meghatározott érték, mely megfelel egy paraméternek. Szinonímája: aktuális paraméter. Ellentétje: *paraméter.*
asszociációs kapcsolat	association	Szemantikai összeköttetés kettő vagy több osztályozó között, mely magában foglalja a példányai közötti kapcsolatokat. Az objektumoknak együttműködésükhöz kapcsolatokkal kell rendelkezniük. A kapcsolatok által nyílik lehetőség arra, hogy az objektumok üzeneteket küldhessenek egymásnak.
asszociációs kapcsolati osztály	association class	Modellezési elem, mely kapcsolat és osztály tulajdonságokkal egyaránt rendelkezik. Egy kapcsolat osztály látható úgy is, mint egy kapcsolat, melynek osztály tulajdonságai vannak, vagy úgy is, mint egy osztály, melynek kapcsolat tulajdonságai vannak.
asszociációs kapcsolati végpont	association end	Egy kapcsolat végpontja, amely a kapcsolatt az osztályozóhoz kapcsolja.
aszinkron hatás	asynchronous action	Egy olyan kérelem, ahol a küldő objektum nem várakozik az eredményekre. Szinonímája: *aszinkron kérelem.* Ellentéte: *szinkron hatás.*
átmenet	transition	Kapcsolat két állapot között, mely azt jelzi, hogy egy objektum az első állapotban végre fog hajtani egy meghatározott feladatot és belép a második állapotba amikor egy meghatározott esemény történik és a meghatározott feltételek teljesülnek.
attributum, tulajdonság	attribute	Az objektumban tárolt információ. Elnevezett hely egy osztályozóban, mely az értékek olyan tartományát írja le, mely az osztályozók példányait képes tárolni. Szinonímája: *attributum.*
azonosítás	identity	Objektum azonosítása névvel, pointerrel vagy egyedi attributummal. Minden objektum egyértelműen azo--nosítható. Két objektum akkor sem azonos, ha állapotaik megegyeznek.
barát	friend	Barátnak deklarálhatók bizonyos külső metódusok, vagy a teljes osztály. Ez az adott osztály bármely adatát, metó-dusát manipulálhatja.
bináris	binary	Két osztály közötti kapcsolat (asszociáció). Az n-akármi

kapcsolat	association	kapcsolat speciális esete.
CRC kártya	Class Resposibility Collaboration card	OO rendszer elemzéskor használt előregyártott űrlap (objektum neve, felelősségek, együttműködő osztályok).
csatolt érték	tagged value	Egy tulajdonság explicit - név-érték típusú - definíciója. A csatolt érték az egyike az UML három kiterjeszthetőségi mechanizmusának. Bizonyos tagok az UML -en belül előre definiáltak, másokat a felhasználó definiálhat.Ld.: *megszorítás, sztereotípus*
csomag	package	Általános célú mechanizmus a logikailag összetartozó modellező elemek (osztályok, használati esetek, stb.) csoportba szervezésére. A csomagok bármilyen modell elemet tartalmazhatnak, és egymásba is ágyazhatók. Egy rendszer gondolkodhat egyetlen magasszintű csomagban, mely mindent tartalmaz, ami hozzá tartozik.
csomópont	node	A csomópont egy futásidejű fizikai objektum, mely egy számítási erőforrást reprezentál, általában memóriával rendelkezik és gyakran számítási képességgel is. Futásidejű objektumok és komponensek tartózkodhatnak a csomópontokon.
csúcs	vertex	Egy forrás vagy egy cél egy átmenethez az állapotgépezetben. Egy csúcspontnak vagy állapota, vagy pszeudo-állapota lehet. Ld.: *állapot, pszeudo-állapot, átmenet.*
delegálás	delegation	Egy objektum azon képessége, hogy válasz üzenetet küld egy másik objektum felé. A delegálás az öröklésheznt alternatíva használható.
destruktor	destructor	Az objektum leromboló metódusa, a megszűnés előtti tevékenységeket végzi el.
diagram	diagram	Modell elemek csoportjának grafikus megjelenítése. Az UML a következő diagramokat támogatja: osztály diagram, objektum diagram, szekvencia diagram, használati eset - diagram, együttműködési diagram, állapot diagram, tevékenység diagram, komponens diagram és telepítési diagram.
dinamikus objektum	dynamic object	Az objektumoknak életciklusuk van (a létrehozástól a megszüntetésig). A dinamikus objektum csak addig foglalja a helyet a memóriában, amíg él. Ellentéte: *statikus objektum.*
dinamikus osztályozás	dynamic classification	Az általánosítás egy szemantikus változata, melyben egy objektum változtathatja típusát vagy szerepét. Ld.: *általánosítás;* Ellentéte: *statikus osztályozás.*
disztribúciós egység	distribution unit	Objektumok vagy komponensek halmaza, melyek mint csoport vesznek részt a feldolgozásban. Egy disztribúciós egység reprezentálható egy futásidejű komponenssel

		vagy aggregáttal.
domén	domain	Olyan ismeret- vagy tevékenységcsoport, melyet az adott szakterület művelői egy meghatározott fogalom- ill. terminológiakörrel jellemeznek és értelmeznek.
egész-rész kapcsolat	whole-part association	Ld.: *tartalmazási kapcsolat*
egyed objektum	entity object	A rendszer lényegi objektuma: valós személy, dolog, hely, fogalom, vagy esemény.
egységbe zárás	encapsulation	Információk (adatok és metódusok) elrejtése egy objektumban úgy, hogy azok csakis az interfészen keresztül, az objektum metódusai által legyenek elérhetők a külvilág számára. Az OO paradigma egyik alapelve.
egyszeres öröklődés	single inheritance	Az általánosítás egy szemantikus változata, melyben egy típusnak csak egy szupertípusa lehet (Egy osztálynak csak egy őse lehet.). Ellentéte: *többszörös öröklődés.*
együttműködés	collaboration	Egy kölcsönhatás definiálása. Ld.: *kölcsönhatás.*
együttműködési diagram	collaboration diagram	Olyan dinamikus objektumdiagram, mely kölcsönhatásokat ábrázol a példányok, ill. az azok közötti kapcsolatok köré szervezve. Egy vagy több üzenet nyomon követése kifejezve az érintett objektumok együttműködését. A szekvencia diagrammal ellentétben az együttműködési diagram a példányok közötti kapcsolatokat mutatja be. A szekvencia diagramok és az együttműködési diagramok hasonló infromációt fejeznek ki, de különböző módon. Ld.: *szekvencia diagram.*
elem	element	Egy modell atomi összetevője.
elemzés	analysis	A szoftverfejlesztési folyamat azon része, melynek elsődleges célja felállítani a problématerület modelljét. Az elemzés arra összpontosít, hogy *mit* kell csinálni, a tervezés arra irányítja a figyelmet, hogy *hogyan* kell csinálni. Ellentéte: *tervezés.*
elemzési idő	analysis time	Olyan dologra utal, amely egy szoftverfejlesztési eljárás elemzési fázisa alatt fordul elő. Ld.: *design time, modeling time.*
ellátó	supplier	Olyan típus, osztály vagy komponens, mely mások által igénybe vehető szolgáltatásokat nyújt. Szinonímája: *Szerver objektum.*
elő- és utófeltétel	precondition / postcondition	Egy operáció specifikálható elő- és utó-feltételeinek megadásával. Ez a rendszer állapotának megadását jelenti az operáció előtt, ill. után. Olyan feltétel, melynek a művelet meghívásakor igaz értéket kell adnia.
erősen típusos	strongly typed	Minden változó-nak deklarálni kell a típusát.

nyelv	language	
érték	value	Egy típustartomány egy eleme.
értelmezetlen	uninterpreted	Fenntartott hely olyan típusok számára, melyek az UML -ben nincsenek meghatározva. Minden értelmezetlen értéknek van egy megfelelő karakterlánc reprezentációja.
esemény	event	Egy meghatározott esemény specifikációja, melynek időbeli és térbeli elhelyezkedése van, és amelyre a rendszer úgy reagál, hogy megváltoztatja valamely objektumának állapotát.(ilyen lehet pl. a billentyűleütés, egérkattintás, hiba- és időzítő esemény). Az állapot diagram kontextusban az esemény olyan dolog, mely képes elindítani (triggerelni) egy állapot átmenetet. Az események fajtái: • helyzeti: pl. amely objektumra kattintottak, az kapja meg az eseményt; • fókuszált: az éppen fókuszban lévő objektum kapja meg az eseményt, neki kell rá reagálnia; • belső: pl. nyomtasd ki, hiba, idő lejárt.
esemény-kezelés	event handle	A rendszeren belül események keletkeznek és terjednek valamilyen szabály szerint. Egy tulajdonos vagy maga kezeli az eseményt, vagy továbbadja tagjainak (message rooting: az üzenet körbejár). Ha egyik objektum sem kezeli le az eseményt, akkor semmi sem történik vele (nem generálódik hibaüzenet).
esemény-vezérelt programozás	event-driven programming	Olyan programozás, mely egy eseménybegyűjtő és szétosztó mechanizmuson alapszik. Az objektumok a beérkezett eseményeket eseménykezelő metódusokkal kezelik.
export	export	A csomag kontextusban egy elem létrehozása kívülről látható csatolt névterülettel. Ld.: *Láthatóság;* Ellentéte: *import.*
fejlesztési folyamat	development process	Részben rendezett lépések halmaza a szoftverfejlesztés során, melyeket adott cél érdekében hajtanak végre. Ilyenek pl. a modell alkotás, vagy a modell implementáció.
felelősség	responsibility	Egy típus vagy egy osztály kötelezettsége. Minden objektumnak megvan a jól meghatározott feladata. Felelős a feladatai elvégzéséért.
felépítmény	architecture	A rendszer szervezeti struktúrája. Egy felépítmény rekurzívan felbontható olyan részekre, melyek interfészeken keresztül egymással kölcsönhatásban állnak, kapcsolatokra, melyek különböző részeket kötnek össze, valamint megszorításokat az összetartozó részekhez.

felhasználások	uses	Egyik használati esettől a másikig terjedő kapcsolat, melyben a viselkedés úgy van definiálva, hogy a korábbi használati eset alkalmazza a későbbi viselkedését.
felhasználói felület	user interface	Az alkalmazás azon része, mely az ember/külvilág és a számítógép közötti kapcsolatot biztosítja (egyedek megjelenítése és adatgyűjtés)
felsorolás	enumeration	Elnevezett értékek listája, melyet egy bizonyos attributum típus tartományaként használnak (pl.: RGBColor={red, green, blue). A logikai típus egy előre definiált felsorolás az {igaz, hamis} értékekkel.
felülírás	override	Nyelvi mechanizmus, ahol az utód osztály metódusa felülírja az ős osztályét.
finomítás	refinement	Olyan kapcsolat, mely egy - bizonyos fokú részletezettséggel specifikált - dolog teljesebb specifikációját reprezentálja. Pl. az osztály tervezése az osztály elemzésének finomítása.
fogadás	reception	Egy osztályozó fogadott jelre való reakciójának deklarálása.
folyamat	process	Olyan szál, mely más szálakkal konkurens módon végrehajtható.
fordítási idő	compile time	Olyan dologra való hivatkozás, mely a szoftver modul fordításakor történik. Ld.: *modellezési idő, futási idő*
forgatókönyv	scenario	Műveletek egy meghatározott sorozata, mely a viselkedéseket illusztrálja. A forgatókönyv illusztrálhat egy kölcsönhatást. Konkrét használati eset (eseménysor), melyre a rendszert használni lehet. A használati eset egy előfordulása.Ld.: *kölcsönhatás.*
formális paraméter	formal parameter	A függvény vagy eljárás fejében a formális paraméterlistában formális paramétereket deklarálhatunk, melyek lokális változókként viselkednek. Aktuális paraméterekről az eljárás és a függvényhívás során beszélhetünk. Az aktuális paramétereket a főprogram (szülő eljárás / procedúra) tölti fel értékekkel, ezért azok az eljárást hívó környezetben érvényes azonosítók. Az aktuális és formális paramétercsere jelenti a szótárat az eljárás hívásakor használatos paraméterek és az eljáráson belüli paraméterek között.Szinonímája: *paraméter.*
függőség	dependency	Kapcsolat kettő vagy több modell elem között, melyben az egyik modell elem (a független) változása hatással van a mások elemre (a függőre).
futás alatti / késői / dinamikus kötés	runtime / late / dynamic binding	Egy metódus futási időben való hozzákötése a program kódjához.

futási idő	run time	Az az időperiódus, mely alatt egy számítógépes program végrehajtódik. Ellentéte: *modellezési idő.*
gyengén típusos nyelv	weakly typed language	A változóknak nem kell típust deklarálni.
használat	usage	Függőségi kapcsolat, melyben egy elem (a kliens) megköveteli egy másik elem (az ellátó) jelenlétét a korrekt működéshez vagy implementációhoz.
használati eset [osztály]	use case [class]	Tevékenységek sorozatának specifikációja, beleértve kül. változatokat, melyeket egy rendszer (vagy más egyed) a rendszer egyéb szereplőjével kölcsönhatásban végrehajthat. Általánosan megfogalmazott eseménysor (eljárás), melyre adott rendszert használati lehet. Ld.: *használati eset példány.*
használati eset diagram	use case diagram	Olyan diagram, mely a rendszerbeli szereplők és használati esetek közötti kapcsolatot ábrázolja. Megmutatja, hogy kik (aktorok), és hogyan használják a rendszert. Aktorok, használati esetek és kapcsolataik.
használati eset modell	use case model	Olyan modell, mely egy rendszer funkcionális követelményeit írja le a használati esetek kifejezésmódjával.
használati eset példány	uses case instance	Meghatározott tevékenységsorozat végrehajtása egy használati esetben. Ld.: *használati eset osztály.*
használati kapcsolat	use association	Ld.: *ismeretségi kapcsolat*
hatás	action	Egy végrehajtható utasítás specifikációja, mely egy számítási eljárás absztrakcióját képezi. Egy hatás a modell állapotának megváltozását eredményezi. Megvalósítása egy objektum felé küldött üzenettel, vagy egy tulajdonságérték (attributum) megváltoztatásával történik.
hatás kifejezés	action expression	Egy kifejezés, mely nem más, mint hatások csoportja (kollekciója).
hátásállapot	action state	Egy olyan hatás, mely egy atomi hatás végrehajtását reprezentálja, jellegzetesen egy művelet meghívása.
hibrid, vegyes OO nyelv	hybrid OO language	Olyan nyelv, melyben objektumorientáltan és hagyományosan egyaránt lehet programozni.
idő	time	Egy érték, mely az időtengely egy abszolut, vagy relatív pozícióját reprezentálja.
idő esemény	time event	Egy esemény, mely azt az időt jelöli, mely a jelenlegi állapot beállta óta telt el. Ld.: *esemény.*
idő kifejezés	time expression	Olyan kifejezés, mely az idő egy abszolút vagy relatív értékét határozza meg.
időzítő jel	timing mark	Egy időre vonatkozó jelölés, melynél egy esemény, vagy üzenet fordult elő. Az időzítő jelek használhatók a megszorításokban.
implementáció	implementation	Annak definíciója, hogy valami hogyan épüljön fel, vagy

		számítódjon ki. Pl. egy osztály egy típus implementációja, vagy egy metódus egy művelet implementációja.
implementáció öröklődés	imlementation inheritance	Egy speciálisabb elem implementációjának öröklődése. Ide soroljuk az interfész öröklődését.
import	import	A csomag kontextusban használt fogalom, olyan függőség, mely megmutatja azon csomagokat, melyek osztályai egy megadott csomagon belülre hivatkozhatnak (beleértve az olyan csomagokat is, melyek rekurzívan beágyazottak).
inicializálás	init	Az objektum - egy erre szolgáló - metódusának meghívása, melynek során az adott osztályba tartozó objektumot életre keltjük. A paraméterként átadott értékek az objektum tulajdonságainak adódnak át (állapota lesz) és végrehajtódnak a működéshez szükséges kezdeti tevékenységek.
interfész	interface	Olyan műveletcsoport deklarációja, melyet az objektum példány által felkínált szolgáltatások definiálásához használhatunk.
interfész objektum	interface object	A külvilággal, a felhasználóval kapcsolatot teremtő objektum.
interfész öröklődés	interface inheritance	A speciálisabb elem interfészének öröklése. Nem tartozik e fogalomhoz az implementáció öröklése.
ismeretségi kapcsolat	acquaintance association	Használati kapcsolat. A vezérlő objektum megszűnése (halála) nem vonja maga után a kapcsolódó objektumok megszűnését. Két objektum ismeretségi (használati) kapcsolatban áll egymással, ha azok léte egymástól független, és legalább az egyik ismeri, ill. használja a másikat.Ellentéte: *tartalmazási kapcsolat.*
jegyzet	note	Megjegyzés, melyet grafikusan a diagramhoz kapcsolhatunk.
jel	signal	A példányok közötti aszinkron kommunikáció stimulusaink specifikációja. A jeleknek lehetnek paraméterei.
jellemző, tulajdonság	feature	Olyan - a művelthez vagy az attributumhoz hasonló - sajátság, mely egy másik egyedbe van beágyazva. Ilyen lehet egy interfész, egy oszály, vagy egy adattípus.
kapocs	link	Szemantikai kapcsolódás objektumok között. Egy asszociáció példány. Ld.: *asszociáció.*
kapocs végpont	link end	Az asszociáció végpont egy példánya. Ld.: *asszociáció végpont.*
karakterlánc	string	A karakterlánc reprezentációjának részletei implementációfüggőek, és magukba foglalhatják a nemzetközi karakterkészleteket és grafikus karaktereket is.
kardinalitás	cardinality	Az elemek száma egy halmazban. Ellentéte: *multiplicitás*.

kérelem	request	A kérelem az objektum példányok felé küldendő stimulus (eseményt kiváltó jel) specifikációja. A kérelem lehet művelet, vagy jel.
keretrendszer	framework	Olyan mikrofelépítmény, mely kiterjeszthető sablont biztosít az alkalmazás számára egy adott alkalmazási területen. Lényegét tekintve egy váz (menü, help, navigálás, stb.), amely minden programban ugyanaz. Ezt kell kiegészíteni az adott alkalmazásra specifikus elemekkel. Egy renszer objektumainak szerkezeti és együttműködési mintája.
kifejezés	expression	Egy karakterlánc, mely egy meghatározott értékként értékelődik ki. Pl. a "(7+5*3)" kifejezés számként értékelődik ki.
kiterjeszt	extends	Kapcsolat a használati esetek között, mely meghatározza a viselkedéseket.
kivételkezelés	exception handling	Egy esemény vagy feltétel, mely megszakítja a program. normális futását. A program ezt átadja az eseménykezelőnek, amely lekezeli.
kliens	client	Olyan osztályozó, mely más osztályozótól kér szolgáltatást. A feladatot elvégeztető objektum. Szinonímája: *kliens objektum.* Ellentéte: *ellátó.*
kód újrafelhasználása	code reuse	Egy már megírt szoftver egység felhasználása (osztály továbbfejlesztése, példány létrehozása).
kölcsönhatás	interaction	Adott feladat végrehajtása érdekében az objektum példányok által egymásnak küldendő üzenetek specifikációja. A kölcsönhatás az együttműködési kontextusban van definiálva. Ld.: *együttműködés.*
kölcsönhatás diagram	interaction diagram	Objektum diagram, mely egy operáció működését írja le az üzenetek időrendi nyomonkövetésével. A diagramon feltüntethetők az épp születő, ill. meghaló objektumok is. Általános kifejezés, melyet számos diagram típusra alkalmaznak. Lényege, hogy az objektumok kölcsönhatásait emeli ki. Ezek közé tartozik: *együttműködési diagramok, szekvencia diagramok, és tevékenység diagramok.*
kommunikációs kapcsolat	communication association	Csomópontok közötti kapcsolat egy telepítési diagramban, mely kommunikációt foglal magába. Ld.: *telepítési diagram*
komponens	component	Szabványos, végrehajtható szoftver építőelem (osztály) azonosítóval és jól definiált interfésszel, melyet az alkalmazások fejlesztésekor használnak fel (pl. interfész, kontroll- és konténer komponensek).
komponens diagram	component diagram	Olyan diagram, mely a komponensek között létesített függőségi viszonyokat és rendezettségeket ábrázolja. A logikai modell osztályainak és egyéb elemeinek fizikai

		csoportosí-tását ábrázolja.
kompozíció	composition	A tartalmazási kapcsolat egy formája erős tulajdonosi viszonnyal és egybeeső élettartammal, a részek az egésszel együtt születnek és halnak meg. A részek multiplicitása nem változhat. A kompozíció lehet rekurzív is. Szinonímája: *kompozit tartalmazási kapcsolat*.
kompozit (összetett) [osztály]	composite	Olyan osztály, mely egy kompozíciós kapcsolat által kapcsolatban áll egy vagy több más osztállyal. Ld. kompozíció.
kompozit állapot	composite state	
kompozit tartalmazási kapcsolat	composite aggregation	Szinonímája: *kompozíció*
konkrét osztály	concrete class	Olyan osztály, melyből közvetlenül létrehozható objektum példány. Ellentéte *absztrakt osztály.*
konkurencia	concurrency	Két vagy több tevékenység előfordulása ugyanabban az időintervallumban. Konkurencia akkor áll elő, amikor egyidejűleg hajtunk végre kettő vagy több vezérlési szálat. Ld.: *vezérlési szál.*
konstruktor	constructor	Az osztály inicializáló metódusa. Feltölti az adatmezőket, elvégzi a kezdeti tevékenységeket, valamint előkészíti az objektumot a virtuális metódusok használatára.
konténer objektum	container object	1. Egy objektum példány, mely más objektum példányok egy kollekcióját tárolja és azokon különböző eléréseket, karbantartási műveleteket biztosít. (pl. láncolt lista, tömb, fa, verem, sor adatszerkezetek). 2. Olyan komponens, mely arra hivatott, hogy más komponenseket foglaljon magába. Ld.: *komponens.*
kontextus	context	A kapcsolatban álló modell elemek halmazának egy nézete olyan különleges célra, mint egy operáció meghatározása.
kontroll objektum	control object	Vezérlést végző objektum.
kötés	binding	Egy modell elem előállítása sablonból a sablon paramétereinek felhasználásával.
követelmény	requirement	Egy rendszer kívánt jellemzője, tulajdonsága vagy viselkedése.
közbülső állapot	substate	Olyan állapot, mely egy kompozit állapot része. Ld.: *konkurrens állapot, független állapot.*

küldés [üzenet]	send [a message]	Üzenet példány küldése a küldő objektumból a fogadó objektum felé. Ld.: *küldő, fogadó.*
küldő	sender [object]	Az az objektum, mely üzenet példányt továbbít egy vevő objektum felé. Ellentéte: *fogadó.*
láthatóság	visibility	Felsorolás, melynek értékei (publikus, védett, privát) azt jelzik, hogy a hivatkozott modell elem csatolt névterülete látható -e a külvilág számára.
logikai	boolean	Felsorolás, melynek értékei: Igaz, Hamis
logikai kifejezés	boolean expression	Olyan kifejezés, melynek kiértékelése logikai értéket ad eredményül.
megjegyzés	comment, remark	Egy elemhez, vagy elemek egy csoportjához fűzött jegyzet, magyarázat, melynek nincsen szemantikája. Ellentéte: *megszorítás.*
megkülön-böztetés	distinction	Objektumok megkülönböztetése a tulajdonságtípusaik alapján. Ld.: *tulajdonságtípus.*
megszorítás	constraint	Modell elemek között szemantikus ösz-szefüggés, szabály szöveges le-írása, feltétel vagy korlátozás. Bizonyos megszorítások az UML -en belül előre definiáltak, másokat a felhasználó definiálhat. A megszorítás az egyike az UML három kiterjeszthetőségi mechanizmusának. Ld.: *csatolt érték, sztereotípia*
meta-metamodell	meta-metamodel	Olyan modell, melyet az UML a metamodell kifejezésére definiál. A meta-metamodell és a metamodell közötti kapcsolat hasonló a metamodell és a modell közötti kapcsolathoz.
meta-osztály	metaclass	Olyan osztály, melynek példányai osztályok (osztályok osztálya). A meta-osztályokat tipikusan metamodellek készítésére használjuk.
metamodell	metamodel	Olyan modell, melyet az UML a modell kifejezésére definiál. A meta-metamodell egy példánya.
metaobjektum	metaobject	Általános kifejezés minden metaegyedhez egy metamodellezési nyelvben. Pl. metatípusok, metaosztályok, metaattribútumok, és metaasszociációk (kapcsolatok).
metódus	method	Egy művelet (operáció) implementációja. Azt az algoritmust, ill. eljárást specifikálja, mely hatással van a művelet eredményére. Általánosságban: egy adott osztályba tartozó objektum működési leírása (eljárás/függvény formájában), amit az osztály definíciójában tárolunk.
minősítő	qualifier	Kapcsolati attributum, vagy attributumok csoportja, melyek értékei részét képezik a kapcsolatban résztvevő objektumok halmazának.
minta	pattern	A fejlesztés alapjául szolgáló sablon (terv, kód, módszer).
modell	model	Egy rendszer szemantikailag zárt absztrakciója. Ld.: *rendszer.*

modell elem	model element	Absztrakció, mely az ábrázolt rendszer egy elemét képezi. Ellentéte: *nézet elem.*
modell szemlélet	model aspect	A modellezés azon dimenziója, mely a metamodell bizonyos jellemzőit hangsúlyozza ki.
modellezési idő	modeling time	Olyan dologra utal, amely egy szoftverfejlesztési eljárás moddellezési fázisa alatt fordul elő. Magába foglalja az elemzési- és tervezési időt. Az objektum rendszerek tárgyalásakor fontos kihangsúlyozni a modellezési idő és a futásidő közötti különbséget. Ld.: *elemzési idő, tervezési idő.* Ellentéte: *futásidő.*
modul	module	A tárolás és a műveletek szoftver egysége. A modul fogalma magába foglalja a forráskód-modulokat, a bináris kód modulokat és a végrehajtható kód modulokat. Ld.: *komponens.*
multiplicitás	multiplicity	A megengedhető számok körének specifikációja, melyet egy halmaz tartalmazhat. Multiplicitás specifikációkat adhatunk pl. az asszociációkhoz, repetíciókhoz és más egyéb célokra. A kapcs. foka. Megmutatja, hogy egy adott halmaz milyen értékeket vehet fel. A multiplicitás egy (esetleg végtelen) részhalmaza a nem negatív egészeknek. Ellentéte: *kardinalitás.*
művelet, operáció	operation	Olyan szolgáltatás, melyet egy objektumtól lehet kérni, és amely hatással van a viselkedésre. Megvalósítása eljárás, függvény, metódus, amit az obj. végre tud hajtani Egy műveletnek szignatúrája van, mely korlátozhatja a lehetséges aktuális paramétereket.
n-bármely kapcsolat	n-ary association	Kapcsolat három vagy több osztály között. A kapcsolat minden egyes példánya egy n-tuple -ja a saját osztálybeli értékeknek.
név	name	A modell elem azonosítására használt karakterlánc.
névterület	namespace	A modell azon része, melyben a nevek definiálhatók és használhatók. A névterületben minden egyes névnek egyedi jelentése van. Ld.: *név.*
nézet	view	Egy modell olyan vetülete (projekciója), mely megadott perspektívából, vagy megfelelő helyzetből szemlélve elhagyja az adott perspektíva szerint lényegtelen egyedeket.
nézet elem	view element	A nézet elem egy szöveges és/vagy grafikus vetülete a modell elemek egy kollekciójának.
nézet vetület	view projection	Modell elemek vetülete (projekciója) a nézet elemekre. A nézet vetület elhelyezést és stílust biztosít minden egyes nézet elem számára.
nyom	trace	Olyan függőség, mely történeti vagy feldolgozási kapcsolatot jelez két elem között. Ugyanazt a fogalmat reprezentálja az egymásból való származtatárukra vonatkozó szabályok specifikációja nélkül.
nyomvonal	swimlane	Elkülönített terület a kölcsönhatás diagramokon a hatásokra kifejtett válaszreakciók szervezésére. Gyakran megfeleltethetők az üzleti modellbeli szervezeti

		egységeknek.
objektum	object	Egyed egy jól definiált határral és azonosítóval, egységbe zárja az állapotot és a viselkedést. Az objektum állapotát az attributumok és a kapcsolatok reprezentálják, a viselkedését a műveletek, metódusok és az állapot gépek. Az objektum az osztály egy példánya. Az objektum lehet: • egyed objektum: hosszabb életű, maradandó információk tárolására alkalmasak, a program lényegi részét tartalmazzák. Ilyenek pl. az adatbázis objektumok. • interfész objektum: gondoskodik az egyedek megjelenítéséről és a külvilággal való kapcsolatról. • kontroll objektum: vezérlést, számolást végrehajtó objektum. Ld.: *osztály, példány.*
objektum alapú	object-based	Bármilyen rendszer, nyelv vagy metódus, mely támogatja az objektum azonosságot, osztályozást, ill. bezárást, de nem támogatja a specializálást, öröklést.
objektum diagram	object diagram	Olyan diagram, mely objektumokat és kapcsolataikat fogja össze egy adott időpillanatban. Az objektum diagram az osztály diagram vagy az együttműködési diagram speciális esetének tekinthető. Ld.: *osztály diagram, együttműködési diagram.*
objektum életvonal	object lifeline	Vonal a szekvencia diagramban, mely az objektum létezését reprezentálja egy adott időperiódusban. Ld.: *szekvencia diagram.*
OO adatbázis-kezelő rendszer	OO database management system (OODBMS)	Olyan adatbázis-kezelő, mely tényleg. objektumokat tárol, és működése OO elveken alapszik.
OO paradigma	OO paradigm	Obj. orientált szemlélet, megközelítési stratégia.
öröklődés (öröklési kapcsolat)	inheritance (inheritance relationship)	Két osztály közötti kapcsolat, melyben az utód osztály örökli az ős osztály tulajdonságait és viselkedését. Olyan mechanizmus, mely által a speciálisabb elemek magukba foglalják az általánosabb elemek struktúráit és viselkedéseit. Egy létező osztálynak a továbbfejlesztése (specializálása: kibővítése/leszűkítése). Lehetővé teszi a kód újra felhasználását. Ld.: *általánosítás*
ős osztály, örökítő	ancestor	Az összes olyan osztály, mely az adott osztály felett helyezkedik el az öröklési ágon. Ellentéte: *utód osztály.*
összefüggési kapcsolat	relationship	Szemantikus kapcsolat modellelemek között. Ilyen pl az asszociációs kapcsolat és az általálnosítás.
osztály	class	Olyan objektumok halmazának a leírása, melyek ugyanazokkal az attributumokkal, operációkkal, metódusokkal, kapcsolatokkal és szemantikával rendelkeznek. Lényegét tekintve egy olyan sablon, mely alapján objektum példányokat hozhatunk létre. Csak a

		viselkedés leírását tárolja, az adatokat nem. Egy osztály az operációk kollekciójának meghatározásához interfészeket használhat. Így biztosítható az osztály környezete. Ld.: *interfész, viselkedés.*
osztály diagram	class diagram	Olyan diagram, mely deklaratív (statikus) modell elemek egy kollekcióját jeleníti meg. Ilyenek lehetnek az osztályok, a típusok, valamint tartalmuk és kapcsolataik. Osztályokat és azok kapcsolatait (társítási, örök-lés) tartalmazó statikus diagram. Tartalmazhat csomagokat, esetleg példányokat is.
osztály-hierarchia diagram	class hierarchy diagram	Csak öröklési kapcsolatokat ábrázoló osztálydiagram.
osztály-könyvtár	class library	Objektum osztályok gyűjteménye.
osztályleírás	class description	Az osztály dokumentációja, az osztály diagramon szereplő osztályok részletesebb kifejtése.
osztályozás	classification	Pontosan ugyanazokat az adatokat tartalmazó és ugyanolyan viselkedésleírással rendelkező objektumok egy osztályba sorolása. Ld.: *viselkedés.*
osztályozó	classifier	Olyan mechanizmus, mely viselkedési és struktúrális jellemzőket ír le. Az osztályozók interfészeket, osztályokat, adattípusokat és komponenseket foglalnak magukba.
paraméter	parameter	Olyan változó specifikációja, mely megváltoztathatja értékét, átadható ill. átvehető. Egy paraméter tartalmazhat nevet, típust és irányt. A paraméterek műveletekhez, üzenetekhez és eseményekhez használatosak. Ld.: *formális paraméter.* Ellentéte: *argumentum.*
paraméterezett (generikus osztály, sablon)	template	Osztály sablon, mely más osztályokkal, objektumokkal és/vagy adatok-kal, operációkkal paraméterezhető. A paramétereket a példány létrehozása előtt meg kell adni. Szinonímája: *paraméterezett elem.*
paraméterezett elem	parameterized element	Egy vagy több kötetlen paraméterrel rendelkező leíró egy oszályhoz. Szinonímája: *sablon.*
példány	instance	Olyan egyed, melyhez bizonyos műveleteket alkalmazhatunk, és melynek meghatározott állapota van. Ez az állapot a végrehajtott műveletek hatásainak függvénye. Az objektum példányt egy adott osztályból hozzuk létre. Az objektum példány adatot tárol (tulajdonság, mezők) és műveletet, feladatot hajt végre (viselkedés, metódus). Ld.: *objektum.*
példány-metódus	instance method	Osztályban szereplő metódus, mely a példányon dolgozik.
példányosítás	instantiation	Példány létrehozása egy adott osztályból.

példányváltozó	instance variable	A példányokban szereplő változók.
primitív típus	primitive type	Előre definiált alaptípus, mint pl. egy egész vagy egy karakterlánc.
projekció	projection	Egy halmaz leképezése saját részhalmazává.
prototípus-készítés	prototype	A rendszer interfészeinek durva megvalósítása, mely a felhasználóval való tárgyalás alapja lehet. Mögötte még nincs igazi program.
pszeudo-állapot	pseudo-state	Egy csúcspont az állapotgépen belül, melynek állapot formája van, de nem viselkedik állapotként. A pszeudo-állapotok magukba foglalják a kezdeti- vég- és történeti vonalakat.
referenci	reference	1. Egy modellelem megjelölése. 2. Elnevezett hely az osztályozóban, mely elősegíti a navigálást más osztályozók felé.
rendszer	system	Összefüggő egységek csoportja, melyek meghatározott cél érdekében vannak egységbe szervezve. Egy rendszer egy vagy több modellel írható le, esetleg több nézőpontból.
repozitori	repository	Tárolóhely objektum modellek, interfészek és implementációk számára.
részesedés	participate	Olyan összefüggés, mely egy objektum példánynak a modell elemben betöltött szerepét mutatja meg.
réteg, szint	layer	Csomagok csoportosításának egy speciális módja a modellben ua. általánosítási szinten.
riport ob-jektum:	report object	Nyomtatott vagy elektronikus listákat készítő objektum.
saját maga	self	A metódus rejtett paramétere, a futó objektum címe.
segédprogram / szolgáltatás	utility	Az osztálynak olyan sztereotípiája, mely globális változókat és eljárásokat csoportosít egy osztály deklaráció formájában. A segédprogram attributumok és műveletek rendre globális változókká és eljárásokká válnak. Egy segédprogram nem alapvető modellezési konstrukció, hanem programozási konvenció.
sokrétűség, sokalakúság	polymorph	Ugyanarra az üzenetre az objektumok különbözőképpen reagálnak attól függően, hogy melyik osztályba tartoznak, vagy milyen az állapotuk. Az üzenet küldőjének nem kell ismernie a fogadó objektum osztályát.
specializálás	specialization	Az adott osztály megkülönböztetése speciális tulajdonságok alapján, ezzel új osztály létrehozása. Az a folyamat, melyben egy objektum leírásához egyedi jellemzőket teszünk.
specifikáció	specification	Valamilyen dolog mibenlétének vagy tevékenységének deklaratív leírása. Ellentéte: *implementáció.*
statikus metódus	static method	Hívása már a fordítónak egyértelmű címet jelent.

statikus osztályozás	static classification	Az általánosítás egy szemantikus változata, melyben egy objektum nem változtathatja típusát ill. szerepét. Ellentéte: *dinamikus osztályozás.*
struktúrális modell-szemlélet	structural model aspect	Olyan modellszemlélet, mely az objektumok struktúráját emeli ki egy rendszeren belül, beleértve a típusokat, osztályukat, kapcsolatrendszerüket, attributumaikat és műveleteiket.
struktúrális tulajdonság	structural feature	Egy modell elem statikus jellemzője, mint pl. egy attributum.
származtatott elem	derived element	Olyan modell elem, melyet más elemből számítanak ki, de vagy az érthetőség kedvéért, vagy tervezési célokból alkalmaznak még akkor is, ha nem nyújtanak szemantikus információt.
szekvencia diagram	sequence diagram	Olyan diagram, mely az objektumok kölcsönhatásainak egymásutániságát mutatja időrendi sorrendben. Konkrét objektumok közötti eseményeket, üzeneteket követi nyomon. Részletesen ábrázolja az objektumok részvételét a kölcsönhatásokban és az egymás között cserélt üzenetek egymásutániságát. Az együttműködési diagrammal ellentétben a szekvencia diagram magába foglal időszekvenciát, de nem tartalmazza az objektumok kapcsolatait. A szekvencia diagram létezhet általános formában (minden lehetséges forgatókönyvet leír) és egy példány formájában (egy aktuális forgatókönyvet ír le). A szekvencia diagramok és az együttműködési diagramok hasonló információt fejeznek ki, de különböző módon ábrázolják. Ld.: *együttműködési diagram.*
szemantikus variáció pont	semantic variation point	A változatok egy pontja a szemantikus metamodellen belül. Szabadságfokot biztosít a metamodell szemantikája számára.
szemétgyűjtés	garbage collection	A hivatkozatlan dinamikus változók automatikus “kisöprése” a memóriából.
szerep	role	Egy egyed elnevezett és specifikált viselkedése egy adott kontextusban. Egy szerep lehet statikus (pl. egy asszociációs kapcsolatbeli szerep) vagy dinamikus (pl. egy együttműködési szerep).
szignatúra	signature	Egy viselkedési tulajdonság neve és paraméterei. Egy szignatúra tartalmazhat opcionális visszatérő paramétert.
szinkron hatás	synchronous action	Egy olyan kérelem, ahol a küldő objektum várakozik az eredményekre. Szinonímája: *szinkron kérelem.* Ellentéte: *aszinkron hatás.*
sztereotípia, sztereotípus	stereotype	A modell elem egy új típusa, mely a metamodell szemantikáját terjeszti ki. Egy csoport tagjának leegyszerűsített véleményét reprezentáló, szabványosított fogalom. Egy bővebb fogalom részhalmaza, fajta (pl. interfész objektum).A sztereotípiáknak bizonyos - a metamodellen belül létező - típusokon vagy osztályokon kell alapulniuk. A sztereotípiák kiterjeszthetik a

		szemantikát, de nem változtathatják meg a létező típusok és osztályok struktúráját. Bizonyos sztereotípiák az UML -en belül előre definiáltak, másokat a felhasználó definiálhat. A sztereotípiák az egyike az UML három kiterjeszthetőségi mechanizmusának. Ld.: *csatolt érték, megszorítás.*
szuperosztály	superclass	Egy másik osztály, az alosztály általánosítása egy általánosítási kapcsolatrendszerben. Az öröklődési ágon az aktuális osztály felett levő osztály. Ld.: *általánosítás.* Ellentéte: *alosztály.*
szupertípus	supertype	Egy másik típus, az altípus általánosítása egy általánosítási kapcsolatrendszerben. Ld.: *általánosítás.* Ellentéte: *altípus.*
tartalmazási hierarchia	containment hierarcy	Olyan hierarchia, mely modell elemekből és a közöttük levő tartalmazási kapcsolatokból áll. A tartalmazási hierarchia egy nem ciklikus gráfot képez.
tartalmazási kapcsolat	aggregation (whole-part association)	A kapcsolat speciális formája, mely egy egész-rész kapcsolatot specifikál az aggregát (egész) és egy komponens rész között. Az egész objektum tartalmazza a rész objektumot. A vezérlő objektum megszűnése magával vonja a kapcsolódó objektumok megszűnését (pl. ha beágyazottan tárolja őket). Ellentéte: *kompozíció, ismeretség.*
tartós objektum	persistent object	Olyan objektum, mely az őt létrehozó folyamat vagy tevékenység szál megszűnése után is életben marad. "Túléli" az alkalmazást. Általában vmilyen háttértárolón tá-rolt obj.
telepítési diagram	deployment diagram	Olyan diagram, mely a futásidejű feldolgozó csomópontok és a komponensek, folyamatok, valamint az objektumok konfigurációját ábrázolja. A komponensek a kódolási egységek futásidejű megnyilvánulásai. A rendszer fizikai eszközeinek és azok kapcsolatainak ábrázolása. Ld.: *komponens diagram.*
termék	artifact, product	Egy információegység, melyet a szoftverfejlesztési eljárás során állítanak elő, vagy használnak. Egy termék lehet egy modell, egy leírás, kód, dokumentáció, munkatervek,vagy egy szoftver.
tervezés	design	A szoftverfejlesztési folyamat azon része, melynek elsődleges célja eldönteni a rendszer implementációját. A tervezés stratégiai és taktikai döntései azért születnek, hogy a rendszerrel szemben támasztott funkcionális és minőségi követelmények teljesüljenek.
tervezési idő	design time	Olyan dologra utal, amely egy szoftverfejlesztési eljárás tervezési fázisa alatt fordul elő. Ld.: *modellezési idő.* Ellentéte: *analysis time.*

típus	type	Osztály sztereotípia, mely az objektum példányok egy tartományának meghatározására használatos az objektumokhoz alkalmazható műveletekkel együtt. Egy típus nem tartalmazhat metódusokat. Ld.: *osztály, példány.* Ellentéte: *interfész.*
típuskényszerítés	type cast	Egy változóra, vagy típusra egy idegen típus "ráhúzása".
típus kifejezés	type expression	Olyan kifejezés, melynek kiértékelése egy vagy több típusra utal.
tiszta OO nyelv	single OO language	Csak objektum orientáltan lehet benne programozni.
többszörös öröklődés	multiple inheritance	Az általánosítás szemantikus változata, melyben egy egy típusnak v. osztálynak több mint egy szülő típusa, ill. osztálya lehet. Ellentéte: *egyszerű öröklődés.*
többszörös osztályozás	multiple classification	Az általánosítás szemantikus változata, melyben egy objektum több mint egy osztályhoz tartozhat. Ld.: *dinamikus osztályozás.*
tranziens objektum	transient object	Olyan objektum, mely csak az őt létrehozó folyamat vagy vezérfonál végrehajtása alatt létezik.
tulajdonság	property	Elnevezett érték, mely egy elem karakterisztikáját jelöli. Egy tulajdonságnak szemantikai hatása van. Bizonyos tulajdonságértékek előre definiáltak az UML -ben, másokat a felhasználó definiálhat.
tűz	fire	Egy állapot átmenet végrehajtása. Ld.: *átmenet.*
ügynök	agent	Olyan objektum, amelyik hol kliens, hol szerver életciklusa során (kér és végrehajt).
újra-felhasználás	reuse	Egy előzőleg már létező termék újbóli felhasználása.
UML, Egységes Modellező Nyelv	UML, Unified Modeling Language	Egyesített Modellező Nyelv. Booch, OMT (Object Modeling Technique), OOSE (Object Oriented System Engineering) egyesítése.
utód / leszármazott [osztály]	descendant	Az összes olyan osztály, amely az adott osztály alatt helyezkedik el az öröklési ágon. Az öröklő osztály.
utófeltétel	postcondition	Olyan feltétel, melynek a művelet befejeződését követően igaz értéket kell adnia.
üzenet	message	Objektum példányok közötti kommunikáció specifikációja. E kommunikáció során az az információ terjed, hogy az egyik példány várakozik a másik példánytól kért művelet/feladat elvégzésére. Az üzenet egy kívülről elérhető metódus hívása.
üzenet küldés	message passing	Az OOP rendszer nem más, mint egymással kommunikáló objektumok összessége. Az objektumok kérik egymást kül. feladatok elvégzésére (kliens-szerver kapcsolat).
vétel [üzenet]	receive [a	A küldő objektum által továbbított üzenet példány

	message]	kezelése. Ld.: *küldő, fogadó.*
vevő	receiver	Olyan objektum, mely a küldő objektum által továbbított üzenet példányt kezeli. Ellentéte: *küldő.*
vezérfonal	thread [of control]	Egy egyszerű útvonal egy program, egy dinamikus modell, vagy a vezérlési folyamat más hasonló reprezentációjának végrehajtása során. Ld.: *folyamat.*
vezérlés fókusza	focus of control	Szimbólum a szekvencia diagramon, mely azt az időperiódust mutatja, melyben egy objektum végrehajt egy tevékenységet közvetlenül, vagy egy alárendelt eljáráson keresztül.
virtuális metódus	virtual method	Olyan metódus, melynek címét a prg. ké-sőbb, futási időben oldja fel. Ezzel elérjük, hogy mindig az aktuális objektum osztályának metódusa fog végrehajtódni.
viselkedés	behavior	Egy operáció vagy esemény megfigyelhető hatásai, beleértve eredményeiket.
viselkedési modell szemlélet	behavioral model aspect	Olyan modellszemlélet, mely az objektum példányok rendszerbeli viselkedését emeli ki, beleértve a példányok metódusait, más példányokkal való együttműködéseit és állapot történeteit.
viselkedési Tulajdonság	behavioral feature	Egy modell elem dinamikus jellemzője, éppúgy mint egy operáció, vagy metódus.