Modellek és adatmodellezés

Modellezési feladatok

Tantárgy:	Intelligens rendszerfelügyelet (VIMIA370)
Szerkesztette:	Micskei Zoltán
Készítették:	Cseppentő Lajos, Darvas Dániel, Hajdu Ákos, Horányi Gergő, Kocsis Imre, Kövi András, Micskei Zoltán, Szatmári Zoltán, Tóth Dániel
Utolsó módosítás:	2015. június 10., verzió: 1.8

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

1 Bevezető

Az *Intelligens rendszerfelügyelet* tantárgy keretében különböző rendszerek modellezési lehetőségeibe is belekóstolunk. Akár egy egyszerű modell elkészítése is nagyon hasznos lehet:

- segít összegyűjteni és megérteni az adott szakterület fogalmait,
- szisztematikus módszert ad, hogy összegyűjtsük az előkerülő fogalmakhoz kapcsolódó szabályokat és kényszereket („egy rendeléshez hány kapcsolattartót lehet megadni?", „kötelező-e kitölteni az értesítési telefonszámot, ha az e-mail meg van adva?" stb.),
- szabványos modellezési nyelvek használatával egyértelműbbé tehetjük, hogy mit értünk az egyes elemeken és kapcsolatokon,
- lehetőség nyílik, hogy automatikus ellenőrzéseket valósítsunk később meg (megadtunk-e minden szükséges adatot, kiszámoljuk a rendszer bizonyos jellemzőjét).

A tantárgy keretében két különböző feladatot néztünk meg részletesebben a félév során:

- *Adatmodellek készítése*: egy adott terület fogalmait és azok kapcsolatát gyűjtjük össze. Tipikusan ez egy magas szintű, kezdeti modell, ami még nem az implementáció közeli részletekre koncentrál.
- *Szolgáltatásbiztonság vizsgálata*: összetett rendszerek rendelkezésre állását, hibatűrését, adott hibajelenségek diagnosztikáját segítjük különböző hibamodellek összeállításával.

A tantárgy vizsgájának gyakorlati részében ilyen feladatok megoldását várjuk el, ez a segédlet a vizsgára való felkészülést segíti. Javasoljuk, hogy a kidolgozott mintapéldákat is először mindenki próbálja *önállóan* megoldani, és csak utána nézze meg a megoldást. A modellezés is egy olyan készség, amit csak gyakorlással lehet fejleszteni, ezért érdemes utána a gyakorló feladatokat is önállóan megoldani (pusztán a megoldás átolvasása még nem elég ahhoz, hogy később alkalmazni is tudjuk az ott látott ismereteket).

1.1 Modellezési alapfogalmak

A modellezés központi fogalom a mérnöki tudományokban, a létező vagy elkészítendő rendszereket modellek segítségével tudjuk megérteni, megtervezni vagy megvalósítani. A modell egy nagyon általános fogalom, valahogy úgy lehetne első közelítésben megfogalmazni, hogy a

modell a „valóság" egy részletének egyszerűsített képe.

Egy modellel kapcsolatban a következő elvárásokat lehet megfogalmazni [2]:

- *Leképezés (mapping)*: a modell egy „eredeti" dolog vagy jelenség leképezése.
- *Csökkentés (reduction)*: az „eredeti" nem minden jellemzője jelenik meg a modellben.
- *Gyakorlati (pragmatic)*: a modell valamilyen szempontból helyettesítheti az „eredetit", használható valamilyen célból.

Az „eredeti" dolog vagy jelenség lehet a valós világ része is, de, mint később látjuk, lehet akár például egy másik modell is.

Modellek létrehozásakor *absztrakció*t használunk. Az absztrakció sokféleképp jelenhet meg: folytonos értékek helyett diszkrét értékeket használunk (pl. pontos távolság helyett csak közel vagy távol megkülönböztetése), sok különböző egyed megkülönböztetése helyett típusok bevezetése (pl. konkrét személyek helyett tanár és diák fogalmak használata), a modellezendő „eredetinek" bizonyos részeit vagy tulajdonságait elhagyhatjuk (pl. számlázó programban az ügyfélnek a neve és a címe fontos, a hajszíne nem) stb.

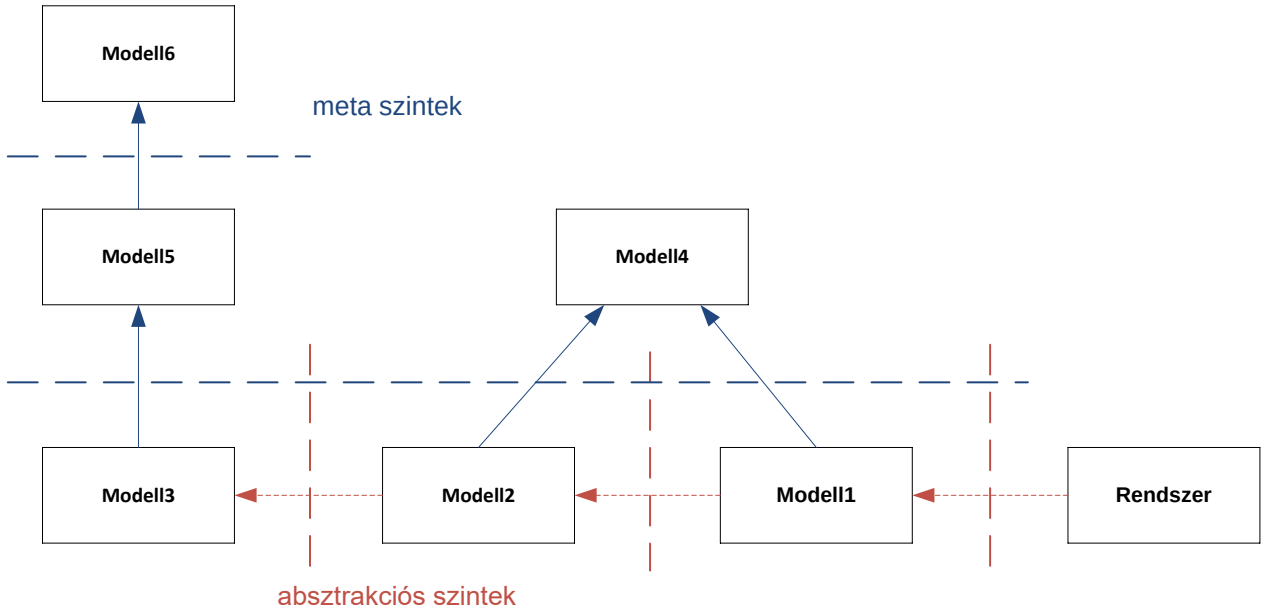
A modelleket nem feltételen kell elkészíteni/lerajzolni/megszerkeszteni, a modell létezhet pusztán csak a fejünkben is. Például amikor egy egyszerű hatványozást elvégző program elkészítése előtt végiggondoljuk, hogy először beolvassuk az „adatot", eldöntjük, hogy „jó" vagy „rossz", elvégezzük a „számítást", majd kiírjuk az eredményt, akkor is tulajdonképpen egy modellt készítettünk.

Azonban néha nem árt leírni a modelleket, például ha együtt akarunk működni másokkal, vagy már nemcsak maga a modellezendő rendszer, hanem a modell is túl bonyolult ahhoz, hogy fejben tartsuk. Ilyenkor tudnunk kell, hogy az adott modell elkészítése során milyen elemeket használhatunk a modellben és azoknak mi a jelentése. Erre való a metamodell:

metamodell: egy modellezési nyelv modellje[1].

A metamodell tulajdonképpen egy sablont ad nekünk, hogy ha ezt az adott modellezési nyelvet akarjuk használni, akkor hogyan nézzen ki egy modell. Megadja például, hogy milyen fogalmak és elemek vannak a modellben, azoknak milyen tulajdonságaik és kapcsolataik vannak, milyen további kényszereket kell teljesíteni (pl. egy B elemhez legfeljebb 3 darab A csatlakozhat). A metamodell és a hozzá tartozó modellek közötti kapcsolatot típusa/példánya (angolul typeOf/instanceOf) kapcsolattal jellemezhetjük[2].

A metamodell készítése és a korábban említett modell készítése azonban két, egymástól független vetület, ezt érdemes mindig fejben tartani. A következő ábra ezt próbálja szemléltetni (1. ábra).



1. ábra: Absztrakciós és metaszintek (a rajzjelek nem szabványosak)

Egy rendszerről készítünk egy modellt (Modell1) valamilyen absztrakció felhasználásával. Ez a modell lehet, hogy túl részletes, amikor később más célból is fel akarjuk használni, így készítünk egy absztraktabb modellt (Modell2) belőle, például elhagyunk bizonyos tulajdonságokat vagy összevonunk bizonyos részeket. Azonban itt még maradunk ugyanannál a modellezési nyelvnél, ezt jelzi, hogy Modell1 és Modell2 metamodellje ugyanaz a Modell4. Később lehet, hogy át szeretnénk térni valamilyen másik modellezési nyelvre, mert az kényelmesebb egy másik feladathoz, így Modell2-ből elkészítjük Modell3-at. Ez már egy másik modellezési nyelvet használ, hisz más a metamodellje. Az ábra mutatja azt is, hogy nem csak két metaszintben gondolkozhatunk, hisz a Modell5-öt is le kell írni valamilyen nyelven, erre szolgál Modell6 (a lánc természetesen nem végtelen, általában legfölül valami olyan

[1] Ez nem teljesen pontos és precíz meghatározás, de ebben a tantárgyban ezzel is tudunk most boldogulni.

[2] Bár van, aki ezt inkább conformsTo/defines kapcsolatnak nevezi [3], ami talán találóbb is. De az instanceOf elterjedtebb, az UML is ezt használja, így most mi is ennél maradunk.

egyszerű metamodell áll, ami például le tudja írni saját magát vagy megelégedtünk a természetes nyelvű leírásával). Természetesen ezt a példát nem csak ebből az irányból lehet „bejárni", kiindulhatnánk a Modell3 magas szintű modellből, és szép lassan finomítással és konkretizációval eljuthatnánk egyre részletesebb modellekig, majd végül az elkészült rendszerig. Például egy szoftver készítése során is használati eseteket veszünk fel, ezek alapján osztálydiagramokat készítünk, majd részletes működést megadó állapotgépeket, végül forráskódot. (Figyelem: a fenti ábra csak szemléltetés, nem szabványos rajzjeleket használ!)

Példa: Az előadás fóliákban szerepel egy részletes példa adatbázisok témaköréből metaszintekről és absztrakcióról, azt érdemes most még egyszer átnézni.

Megjegyzések (ezeket elsőre át is lehet ugrani, érdeklődök gondolkozzanak esetleg el rajta később):

- Felmerülhet, hogy mi is pontosan a jobboldalt álló rendszer, kell-e ott egyáltalán rendszernek állnia. Például mi a helyzet, ha egy programot készítünk, olyankor minek tekinthető a program binárisa vagy a futó példánya?
- Igazából a metamodell készítése is tekinthető egyfajta absztrakciónak, hisz osztályozást (classification) végzünk. Jobb elnevezés híján most maradunk az absztrakciós szinteknél, amikor egyik irányú és metaszinteknél, amikor a másik irányú mozgásra hivatkozunk.

Természetesen mindenki dolgozhat ki saját magának modellezési nyelveket (sőt, az úgynevezett szakterület-specifikus modellezési nyelvek, angolul domain-specific modeling languages, elterjedésével ez egyre gyakoribb), azonban ez akkor csak akkor hasznos, ha precízen megadjuk a nyelvet. Ha egy grafikus modellező nyelvet készítünk, akkor az alábbiakat érdemes elkülöníteni:

- *absztrakt szintaxis*: a nyelv elemkészletét és azok kapcsolatát definiálja,
- *konkrét szintaxis*: a nyelv elemeinek grafikus jelöléseit kapcsolja az absztrakt szintaxis elemeihez,
- *jólformáltsági kényszerek*: megkötéseket adnak, hogy mikor kapunk helyes modelleket,
- *szemantika*: a nyelv elemeinek jelentését adják meg, hogy mit fejez ki egy adott modell.

Például ha a digitális technikában megismert véges automatákat vesszük, akkor ehhez a modellezési nyelvhez az absztrakt szintaxis megadja, hogy olyan elemeink vannak, hogy „állapot" és „átmenet", valamint, hogy az állapothoz kapcsolódhat átmenet. A konkrét szintaxis definiálja, hogy az „állapotot" körrel jelöljük, az „átmenetet" pedig nyíllal. Egy jólformáltsági kényszer lehet, hogy kell pontosan egy kezdőállapotot kijelölni. A szemantika pedig megadja, hogy az „állapotok" a modellezett rendszer állapotait, működési módjait definiálják, a modell dinamikus működése pedig az, hogy egyik állapotból átmehetünk a másikba átmenetet megadott esemény hatására.

1.2 Adatmodellek készítése UML segítségével

Adatmodellek[3] készítése során a következő folyamatot szoktuk végrehajtani:

1. Az adott terület fontos fogalmainak összegyűjtése.
2. A fogalmak az adott modell szempontjából fontos tulajdonságainak meghatározása, a fogalmak közötti kapcsolatok definiálása.
3. Modell elkészítése a kiválasztott modellezési nyelvben.
4. Példánymodellek készítése, amik visszacsatolásként szolgálhatnak (pl. mire van még esetleg szükség a modellben, kényelmesebb lenne egy másik fajta absztrakciót használni…).

Többféle leírási formát használhatunk adatmodellek készítésére, mi most a tárgy keretében UML osztálydiagramokat alkalmazunk.

[3] Az ilyen modellekre szoktak még domain model vagy concept model néven is hivatkozni, mi most adatmodellnek nevezzük a tárgyban.

EA: A felkészüléshez első lépésként nézzük át a modellezés előadás anyagát (legyünk tisztában az UML alapvető elemkészletével és azok jelentésével).

A modellezés nem egy egzakt folyamat, egy adott környezethez sokféle modellt lehet készíteni (pl., egy adott tulajdonságot attribútumként jelenítünk meg vagy külön osztályban ábrázoljuk, vagy milyen mértékben használunk öröklést stb.). Ezért a segédletben megadott megoldásokhoz képest természetesen más megoldások is elképzelhetőek. Ezeknek a „jóságát" nehéz definiálni, az lehet szempont, hogy tartalmaz-e minden megadott adatot, mennyire könnyű később bővíteni, mennyire kényelmes használni, mennyire egyértelmű stb.

Tipikus hibák és általános tanácsok:

- Magas szintű adatmodell készítése esetén nem kell a kapcsolatnak megfelelő attribútumokat felvenni (tehát pl. arraylistek a kapcsolódó osztályoknak megfelelően), ez annál absztraktabb modell. Most még ne programozási nyelveken való megvalósításban gondolkodjunk, hanem fogalmakban és kapcsolatokban.
- Egy adatmodellbe túl sok értelme nincsen interfészeket berakni, főleg olyat, aminek nincs egy metódusa sem (attribútumot meg eleve nem illik interfészbe rakni). Használjunk helyette inkább absztrakt osztályokat.
- UML példány modell készítése esetén már nem szokás a kompozíciót berajzolni, csak sima vonallal jelöljük az objektumok közötti linkeket (bár sok UML modellező eszköz kompozíciót használ példány szinten is).
- A modellezés feladatnál érdemes elolvasni a teljes feladat szövegét, mert ha meg van adva egy konkrét környezet, amit utána példány modellként el kell készíteni, az sokat segíthet.
- Érdemes valami egységes elnevezési koncepciót használni, osztálynévben PascalCase, példány névben tipikusan camelCase formát használunk. Lehetőleg ne használjunk ékezetet vagy szóközt modell elemek nevében, az csak feleslegesen megnehezíti később a feldolgozását.
- Mindig legyen a példányoknak egyértelmű neve, azzal lehet azonosítani őket.

1.3 További információ

[1] Kirill Fakhroutdinov. UML Diagrams. website, URL: http://www.uml-diagrams.org/

Jó webes összefoglaló az UML-ről, sok példával

[2] J. Ludewig. „Models in software engineering – an introduction". Software and Systems Modeling 2(1), 2003, pp. 5–14. DOI: 10.1007/s10270-003-0020-3

Egy olvasmányosabb cikk arról, hogy mi a szerepük a modelleknek szoftver rendszerekben

[3] Jean Bézivin. "On the unification power of models". Software and Systems Modeling 4(2), 2005, pp. 171–188. DOI: 10.1007/s10270-005-0079-0

Tudományos cikk modellekről, metamodellekről

2 Kidolgozott mintapéldák

Ebben a fejezetben korábbi vizsgapéldák szerepelnek megoldásokkal együtt.

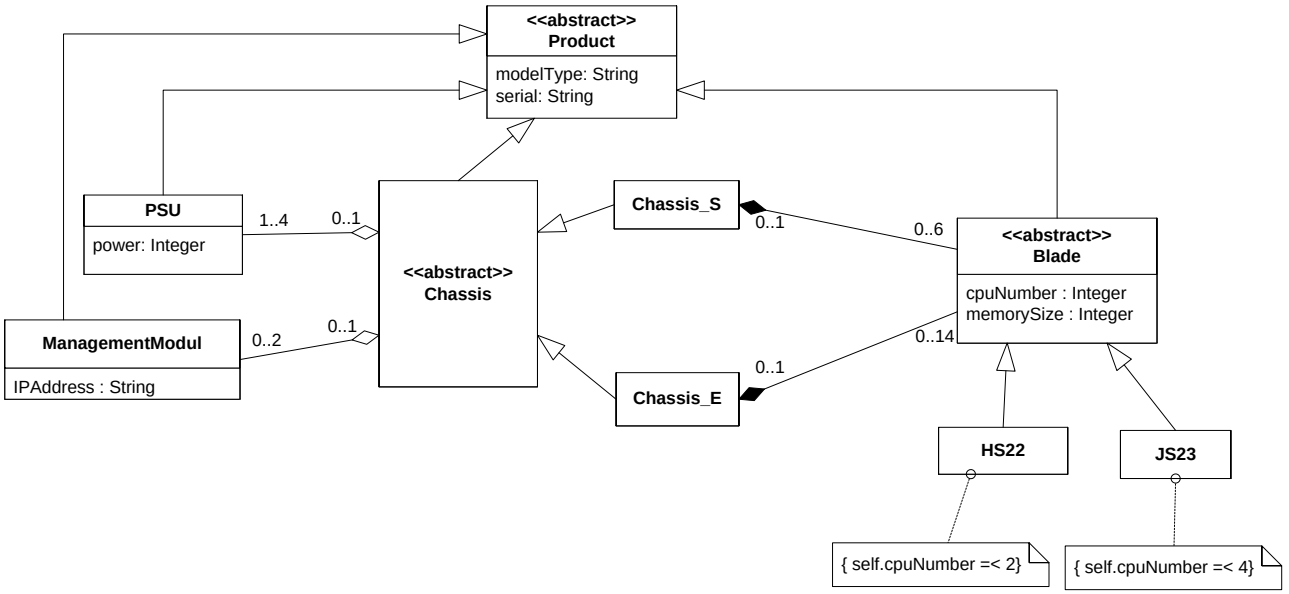
2.1 IBM BladeCenter

2.1.1 A feladat szövege

1. IBM BladeCenter rendszerek modellezéséhez készítsen egy egyszerű metamodellt, melynek segítségével a következő adatokat tudjuk majd tárolni. Egy BladeCenter rendszer egy keretből (chassis) áll, amibe penge szervereket (blade) lehet berakni. Jelenleg E és S típusú keretekkel foglalkozunk, az E-be 14 darab, az S-be 6 darab penge fér. A kereteket és pengéket az IBM a modell számukkal azonosítja, az egyes konkrét termékeknek pedig egyedi sorozatszáma van. A keretekbe a pengéken kívül kell még tápegység (maximum négy fér egy keretbe, különböző teljesítményű modellek kaphatóak) és legfeljebb kettő úgynevezett menedzsment modul. A menedzsment modulon keresztül lehet távolról felügyelni a keretet, a modult ilyenkor IP címével érjük el. A pengékről tárolni akarjuk a bennük lévő fizikai CPU-k számát és a memória méretét. Két féle pengét akarunk jelenleg nyilvántartani, a 4 CPU foglalattal rendelkező JS23-ast és a két CPU foglalatos HS22-est. (6 pont)
2. A fenti metamodellhez készítsen el egy példánymodellt. Egy 8677-3TG modellű E-s keretet vettünk az eBay-en. A keret két 74P4452 típusú 2000 wattos tápegységgel és egy menedzsment modullal érkezett, a modult még nem állítottuk be. A modul sorozatszáma 11373P92. A keret egy darab pengével érkezett, egy 7996-60 típusú JS23-assal, amiben 2 processzor és 64 GB memória van. A modellben jelölje a hiányzó adatokat is, amiket még ki kéne tölteni a metamodell alapján. (4 pont)

2.1.2 Egy lehetséges megoldás

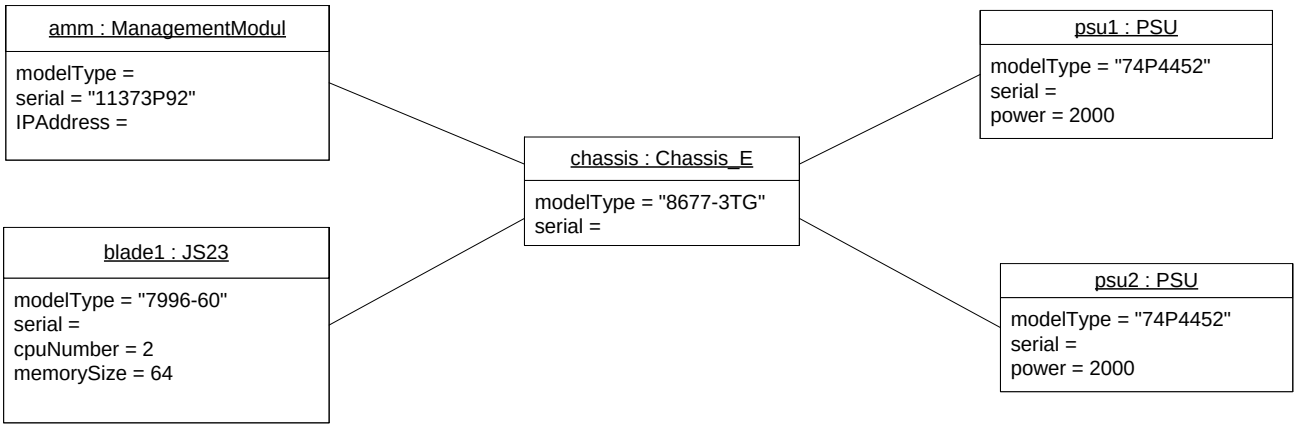
1. feladat



Megjegyzések és tapasztalatok:

- Bevezettünk egy absztrakt ősosztályt, hogy a mindenkinél szereplő modell- és szériaszámot egyszerű legyen kezelni.
- Figyeljük meg, hogy ez egy magas szintű modell, tehát pl. láthatóságot nem feltétlen kell bele rakni.
- A konkrét pengék maximális CPU száma itt OCL kényszerek (constraint) segítségével van jelezve. Ez lehetne akár egy sima megjegyzés is jelen esetben, vagy például az attribútum típusát felül lehetne definiálni a leszármazott osztályban egy megfelelő értéktartományú típussal.
- A tápegységek (PSU) számossága itt most 1..4, de akár lehetne 0..4 is (mindkettő mellett lehet érvelni).
- A chassis-blade kapcsolat reprezentálása volt még érdekes a modellben. Sima asszociáció esetén ugye az a gond, hogy akkor lehetne, hogy egy Blade példány egyszerre tartozzon egy S és ez E kerethez is. Ezért itt kompozíciót (composite aggregation) használtunk, ami az UML-ben egy erősebb fajtája az aggregációnak, egyszerre csak egy kompozícióként jelölt kapcsolatban szerepelhet egy példány.
- Sokadszorra előkerült, de újfent érdemes kihangsúlyozni, hogy interfészt ne nagyon rakjunk adatmodellbe. Egy interfész attól interfész, hogy metódusai vannak (különösen problémás dolog attribútumokat rakni egy interfészbe). Adatmodellben absztrakt osztályokat használunk.
- Fontos, hogy az attribútumnak legyen típusa. Ehhez használjuk az UML általános típusait (String, Integer, Boolean…), és ne valamelyik programnyelv speciális implementáció közeli típusát (pl. int32).
- Az asszociációkat és az asszociáció végeket el lehet nevezni. Mivel most itt a legtöbb szerep egyértelmű volt, ezért ettől eltekintettünk a modell megalkotása során.

2. feladat: Itt megint az a lényeg, hogy a saját magunk által megadott metamodellnek típushelyes példánya legyen a modell, és ki lehessen fejezni a példában szereplő elemeket.



- Arra figyeljünk, hogy az UML példány szinten már nem szokta jelölni az aggregációt vagy kompozíciót, ott már csak sima kapcsolatok (link) vannak.
- Ugyanúgy nem lehet már multiplicitást sem megadni a kapcsolatokon. Ha valamiből két példányom van, akkor azt két külön, különböző nevű objektummal kell ábrázolni.
- A példány neve elhagyható (bár nem javasolt), azonban a kettőspontot és mögötte a típusnevet kötelező megadni, ettől tudom, hogy az minek a példánya.
- Ha egy attribútumnak nincsen értéke, akkor hagyjuk üresen az érték részt, és ne „???”-et írjunk oda.

2.2 SharePoint alkalmazások modellezése

2.2.1 A feladat szövege

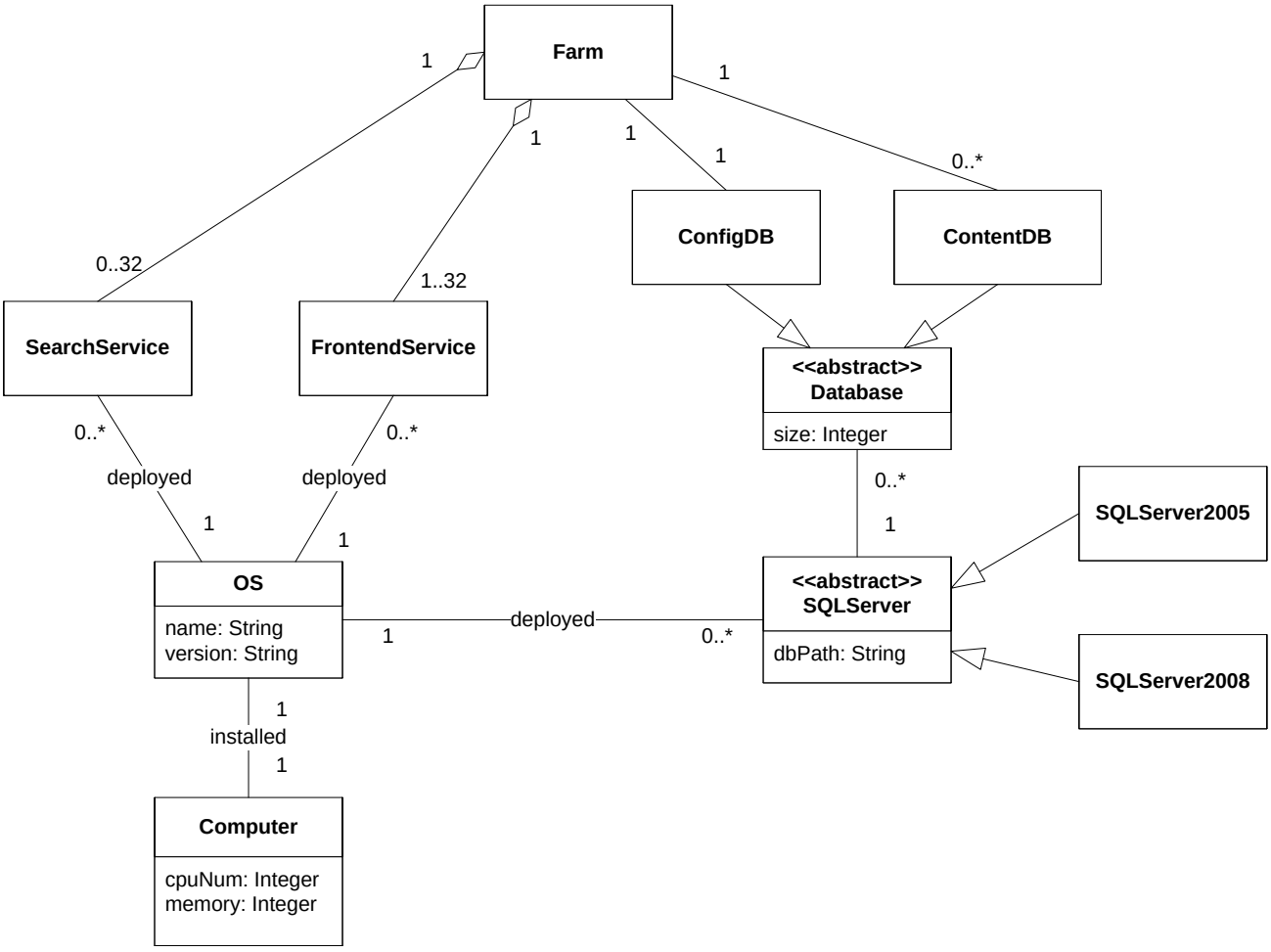
a) Microsoft SharePoint platformra fejlesztünk alkalmazásokat, és a fejlesztői és teszt rendszerekhez használt infrastruktúrák modellezéséhez kell egy metamodellt készítenünk. A SharePoint flexibilis telepítési opciókat ajánl. A telepítés alapeleme a farm. Egy farm működéséhez legalább egy web frontend szolgáltatás kell, és opcionálisan lehet kereső szolgáltatást is telepíteni. A web frontend és keresés telepíthető ugyanarra a számítógépre, ezekből a szerepekből külön-külön legfeljebb 32 lehet a farmban. A modellben tárolni szeretnénk, hogy melyik szolgáltatás melyik számítógépre van telepítve, azon milyen operációs rendszer van (annak mi a verziója), valamint, hogy a számítógépben hány processzor és mennyi memória van. A farm működéséhez ezen kívül szükség van az adatokat tároló adatbázisokra. Pontosan egy darab konfigurációs adatbázis kell, és tetszőleges sok tartalom adatbázist adhatunk meg. Az adatbázisokról tudni akarjuk a méretüket. Az adatbázisokat SQL Server 2005 és 2008-on tárolhatjuk, az adatbázis szerverről az alapértelmezett adatbázis elérési útvonalat jegyezzük fel. A metamodellben figyeljünk a multiplicitások jelölésére! (6 pont)

b) Készítsünk egy példány modellt a fenti metamodellhez. Egy közepes méretű tesztrendszerünk van. A farm két frontend szerverből áll, az egyikre telepítve van a kereső szolgáltatás is. Ezen kívül van egy SQL 2008 adatbázis szerverünk, melyen a 100 MB-os konfigurációs adatbázison kívül egy 500 MB-os és egy 3 GB-os tartalom adatbázis van. Az adatbázis szerver egy négyprocesszoros, 32 GB-os, a két frontend pedig egy-egy kétprocesszoros, 8 GB memóriával rendelkező gép. (3 pont)

c) Módosítsuk az a) feladatban elkészített metamodellt úgy, hogy jelölni tudjuk, hogy ha a szolgáltatásokat virtuális gépekre telepítjük. Rajzolja le külön a metamodell megváltozott részét! (1 pont)

2.2.2 Egy lehetséges megoldás

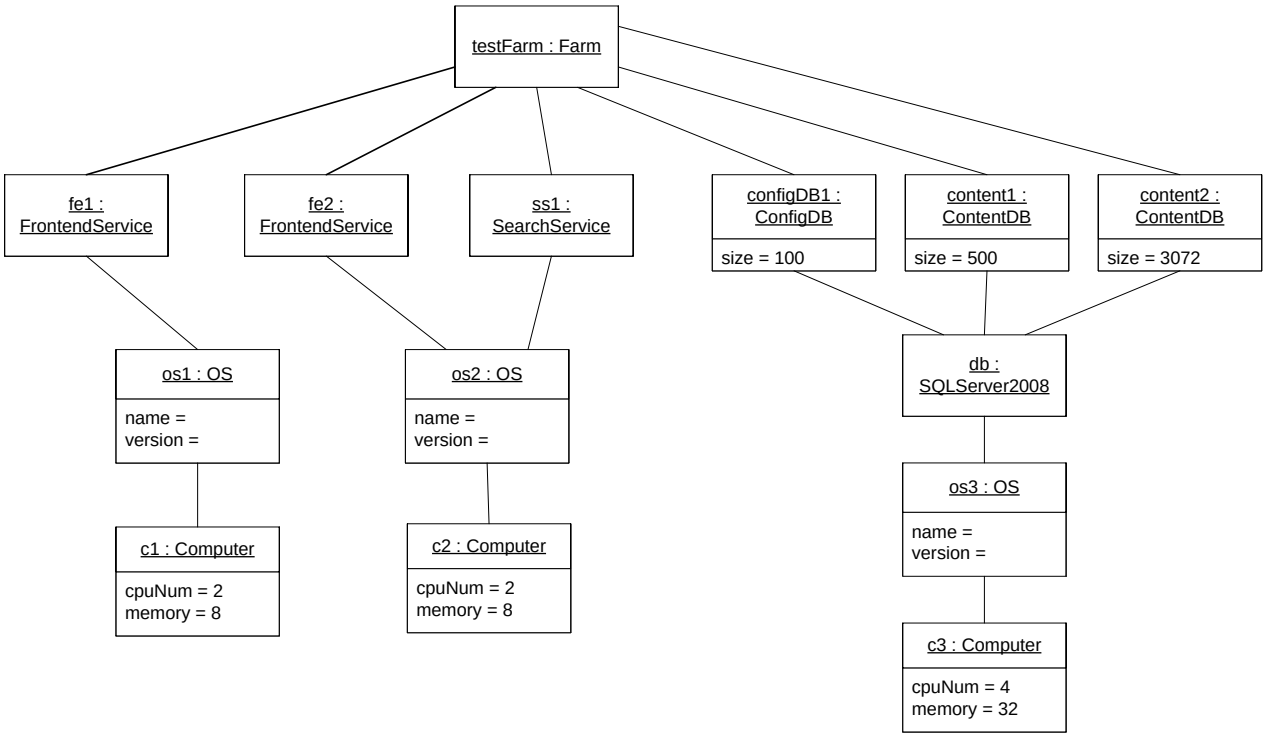
a)



Persze sok mindent lehet kicsit másképp modellezni:

- Az operációs rendszer lehet a számítógép attribútuma.
- A különböző adatbázis típusok lehetnek nem külön osztályok, hanem indulhat a farmból két különböző elnevezett asszociáció.
- A különböző szolgáltatásoknak is lehet egy absztrakt ősosztályt készíteni.
- Az SQL szerver típusát lehet egy megfelelő enumeráció típusú attribútummal megadni.
- A farm és a szolgáltatások között mehet sima asszociáció is, bár az aggregáció talán szerencsésebb.
- …

b) Itt a lényeg, hogy az a)-ban megadott metamodell típushelyes példánya legyen a megadott modell.



c) Itt is sokféle lehetőség van:

- isVirtual attribútum felvétele a Computer osztályba
- VirtualMachine osztály bevezetése, ami bekerül az OS és a Computer közé
- …

2.3 Virtualizációs menedzsment alkalmazás

2.3.1 A feladat szövege

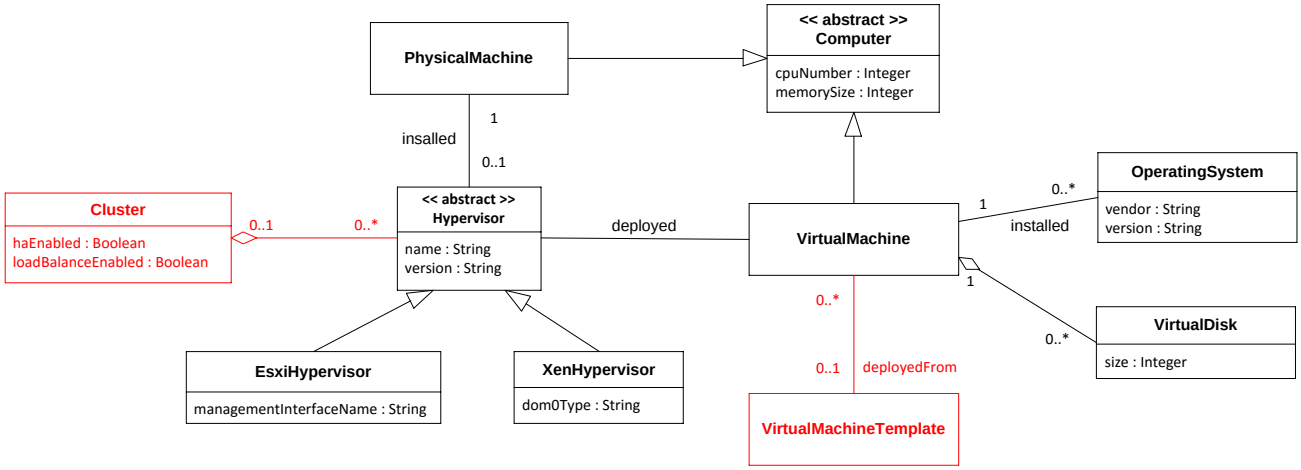
a) Szeretnénk egy saját alkalmazással betörni a virtualizációs piacra, amivel hypervisorokat és virtuális gépeket lehet platformfüggetlenül menedzselni. Ehhez viszont először át kéne látni, hogy milyen fogalmakkal kell dolgozni. Készítsünk tehát egy UML modellt, ami a szakterület legfontosabb elemeit áttekinti. Vannak hypervisor megoldásaink, amikről a verziójukat és a nevüket akarjuk tárolni. Jelenleg két implementációt támogatunk (VMware ESXi és Xen), ESXi esetén azt kell még tudni, hogy mi a menedzsment interfész neve, Xen esetén pedig a dom0-ban futó operációs rendszer típusát. A rendszerben ezen kívül vannak virtuális gépeink, amik valamilyen operációs rendszert vagy rendszereket futtatnak (az operációs rendszert a gyártó és a verzió azonosítja), továbbá valamelyik hypervisor példányon futnak. A hypervisorok valamilyen fizikai gépre vannak feltelepítve. A fizikai és virtuális gépekről egyaránt a processzorok számát és a memória méretét akarjuk nyilvántartani. A virtuális gépekről tárolni kell továbbá, hogy hány és mekkora virtuális lemez tartozik hozzájuk. (6p)

b) Készítsünk egy példány modellt a fenti metamodellhez, amiben legalább két hypervisor és három darab virtuális gép van. (2p)

c) Az alkalmazás új verziójában már a haladó funkciókat is támogatni kell, egészítsük ki a modellt ennek megfelelően. Virtuális gépeket lehet sablonból létrehozni. A hypervisorokat lehet fürtökbe szervezni, ilyenkor opcionálisan be lehet kapcsolni a hibatűrési vagy erőforrás-kiegyenlítési funkciókat a fürtön. (2p)

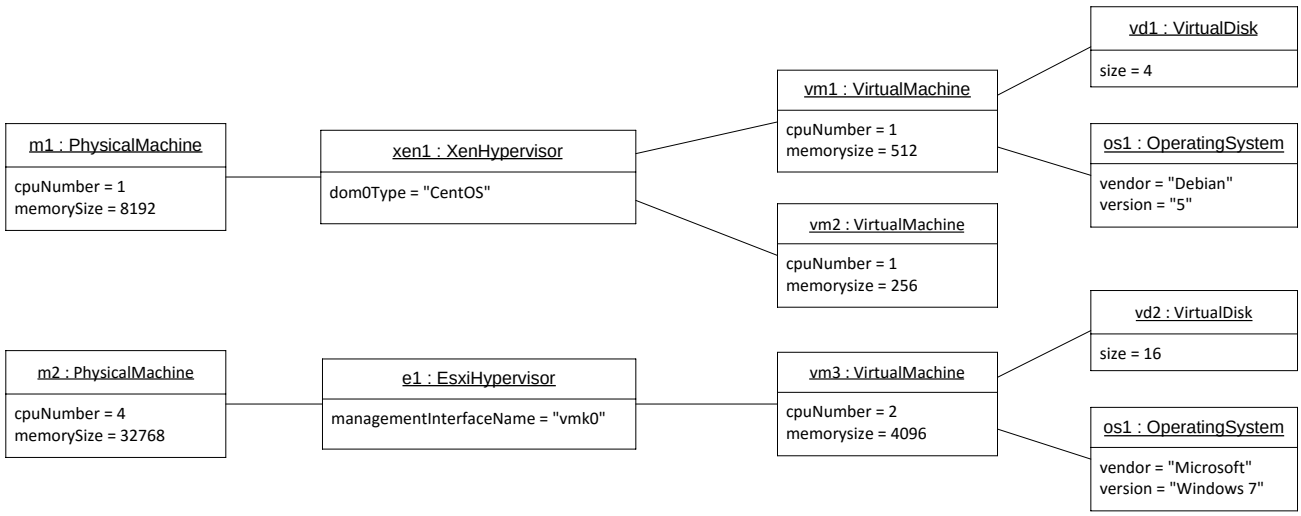
2.3.2 Egy lehetséges megoldás

a)



- Ez a modell csak azt engedi meg, hogy a fizikai gépekre hypervisort telepítsünk, „klasszikus” operációs rendszert nem. A feladatban ez a rész nincs egyértelműen definiálva, lehetne az OperatingSystem az absztrakt Computer osztályhoz is csatolva.
- A fizikai gépre legfeljebb egy hypervisort telepíthetünk a fenti modell szerint. Ha multi-boot konfigurációkat is kezelni akarunk, akkor a multiplicitás lehet 0..* is.
- A XenHypervisor esetén a dom0Type is egy sima String, de akár lehetne egy attribútum helyett ez egy, az OperatingSystem osztályra mutató kapcsolat.

b)



A lényeg itt is annyi, hogy az általunk készített modell típushelyes példánya legyen.

c)

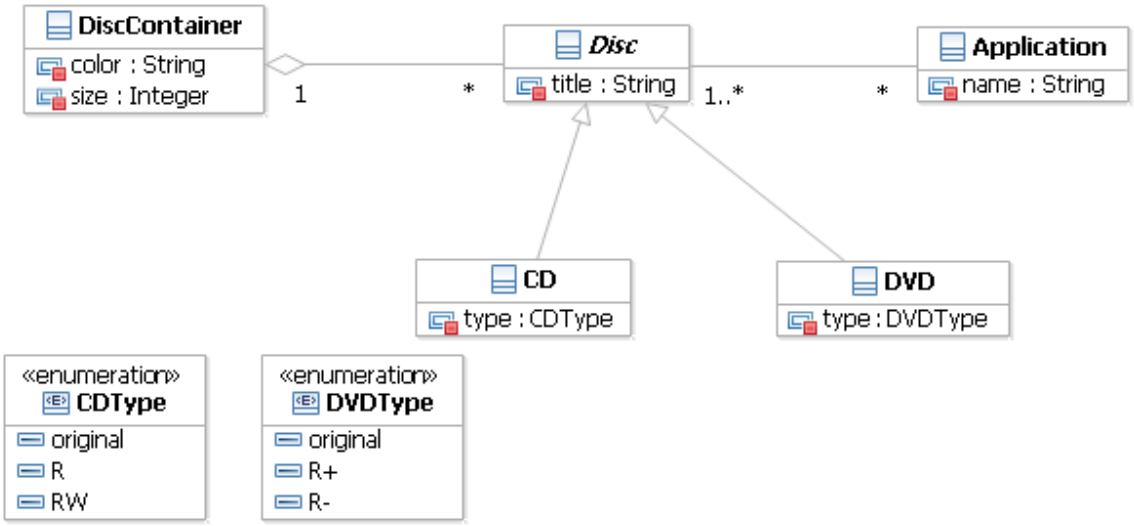
Az a) feladatban pirossal rajzolt kiegészítések.

3 Gyakorló feladatok

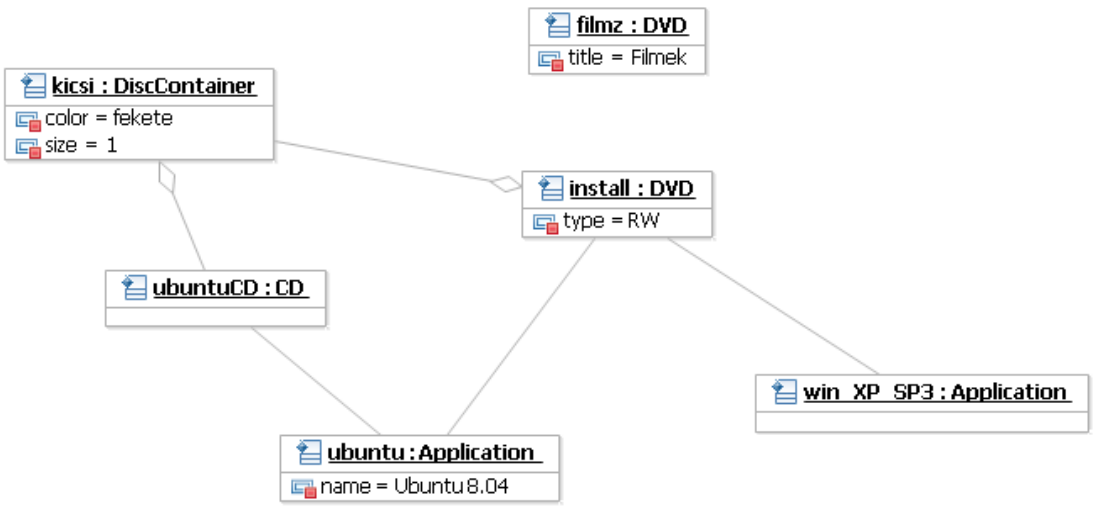
A következő fejezet rövidebb modellezési példákat tartalmaz, melyek segítenek az alapok elsajátításában.

3.1 CD tárolás

Hogy átlássuk a nagyüzemi CD és DVD írás beindulása óta kialakult káoszt, a lemezekről és tárolókról a következő információkat tartjuk nyilván.



Típushelyes példánya-e a fenti UML osztálydiagramon megadott metamodellnek a következő objektumdiagramon lévő modell?



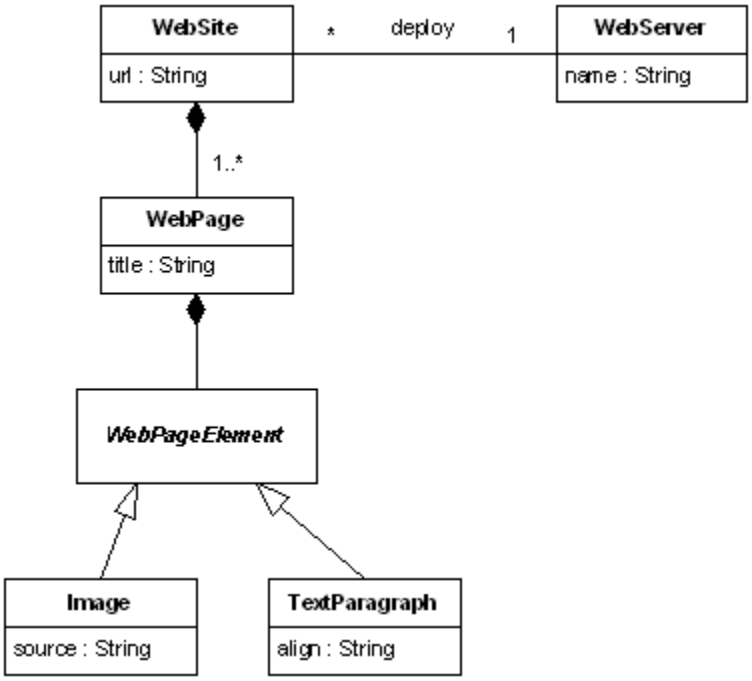
3.2 Fájlrendszer jogosultságok

Készítsünk egy olyan UML osztálydiagrammal megadott metamodellt, mellyel a fájlokra vagy könyvtárakra beállított fájlrendszer jogosultságokat lehet leírni! Minden elemhez egy jogosultsági listát lehet rendelni. A lista egy eleme egy entitásból (felhasználó vagy csoport) áll, akire a jogosultság

vonatkozik, és egy jogosultságból áll. A lehetséges jogosultságok a következőek: nincs hozzáférés, olvasás, írás és teljes hozzáférés.

3.3 Webhelyek ábrázolása

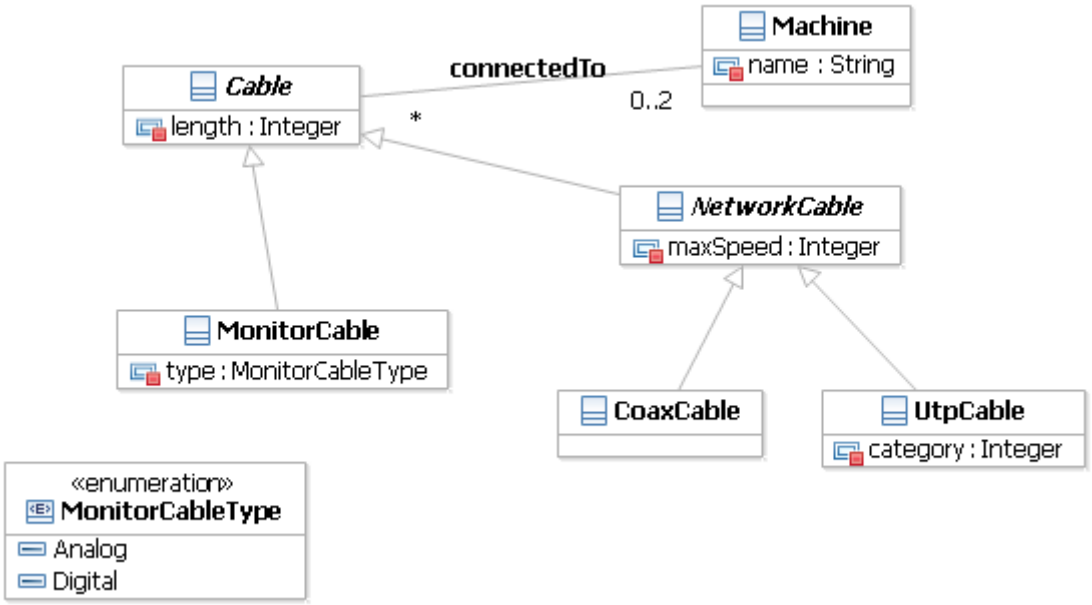
Adott a következő, webhelyeket leíró UML osztálydiagram:



Készítsünk el egy olyan UML objektumdiagram példányát ennek, ami egy két lapból álló webhelyet ábrázol, amiben minden oldalon legalább két elem van.

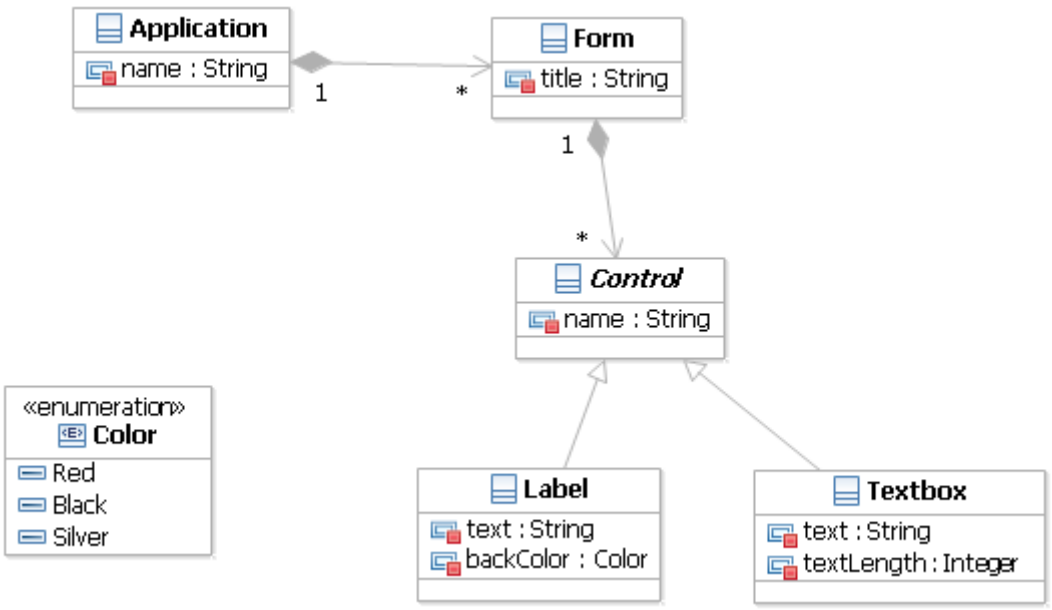
3.4 Hálózati kábelek

Rendet kéne rakni a fiókokban elfekvő és a számítógépekbe bekötött rengeteg számítógépes kábel között. Van két gépünk, mindegyik bekötve a hálózatba és monitorral ellátva, és a rendszergazda úgy tippeli, hogy a raktárban még van legalább egy pót monitorkábel és két hosszabb UTP kábel, sőt még mintha egy koax kábel is maradt volna a hőskorból. Készítsünk mindezek dokumentálására egy UML objektumdiagramot az alábbi UML osztály diagramnak megfelelően!

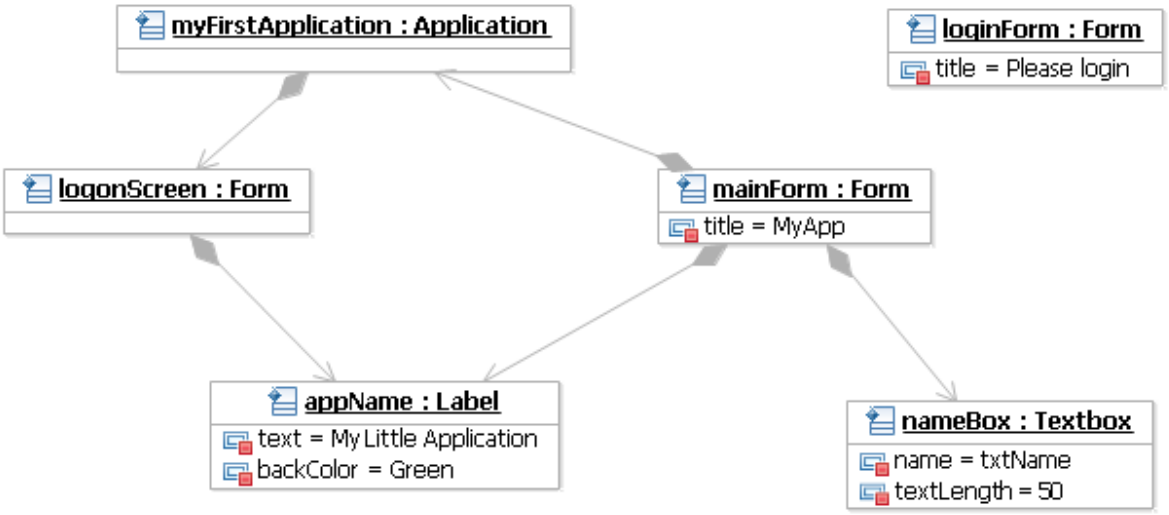


3.5 UI modellezés

A programunk felületének leírásához a következő metamodellt használjuk.



Sikerült-e a fenti UML osztálydiagramon megadott modellnek egy típushelyes példányát megalkotnunk a következő objektumdiagramon?

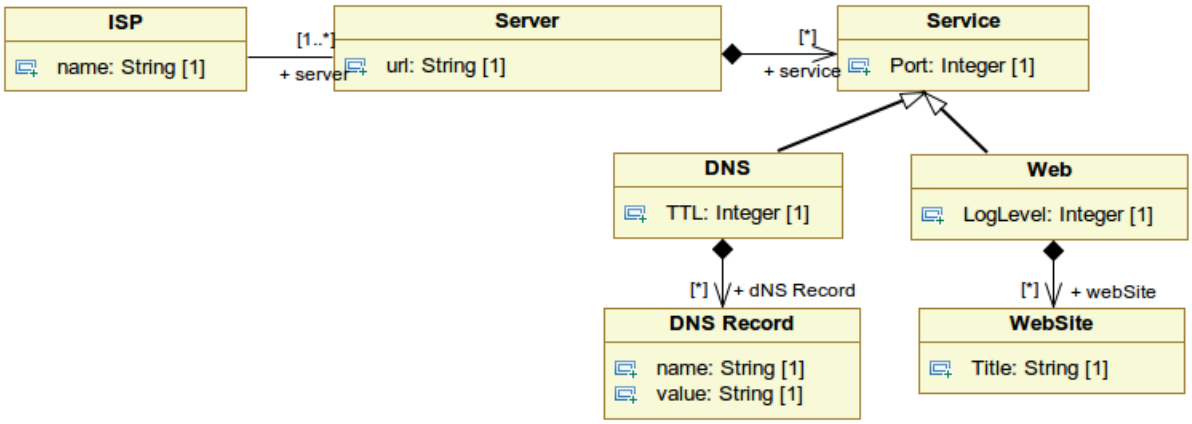


3.6 Alkalmazások nyilvántartása

Készítsen egy olyan metamodellt és ábrázolja egy UML osztálydiagramon, ami számítógépre telepített alkalmazásokat tart nyilván. Az alkalmazásokhoz megadható a nevük és a verziójuk, valamint, hogy a számítógép melyik meghajtójára telepítettük (a meghajtókat a betűjelükkel azonosítjuk). Tároljuk továbbá, hogy melyik alkalmazásnak ki a gyártója, és mi a gyártó weboldala. A kapcsolatoknál ábrázolja azok számosságát is!

3.7 Internetszolgáltatók

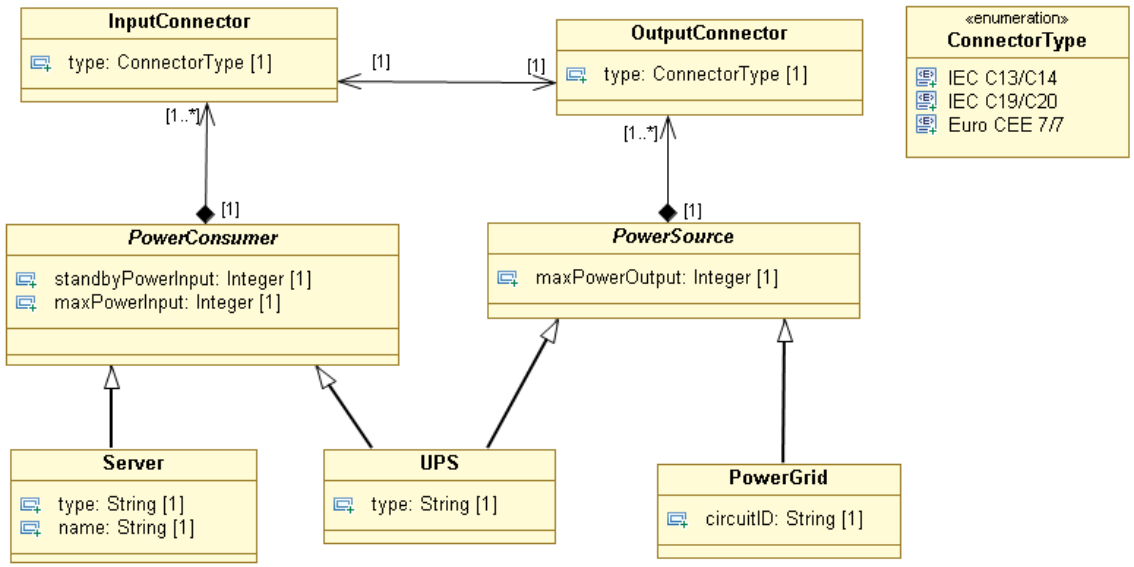
Adott a következő, internet szolgáltatót és szolgáltatásait leíró UML osztálydiagram:



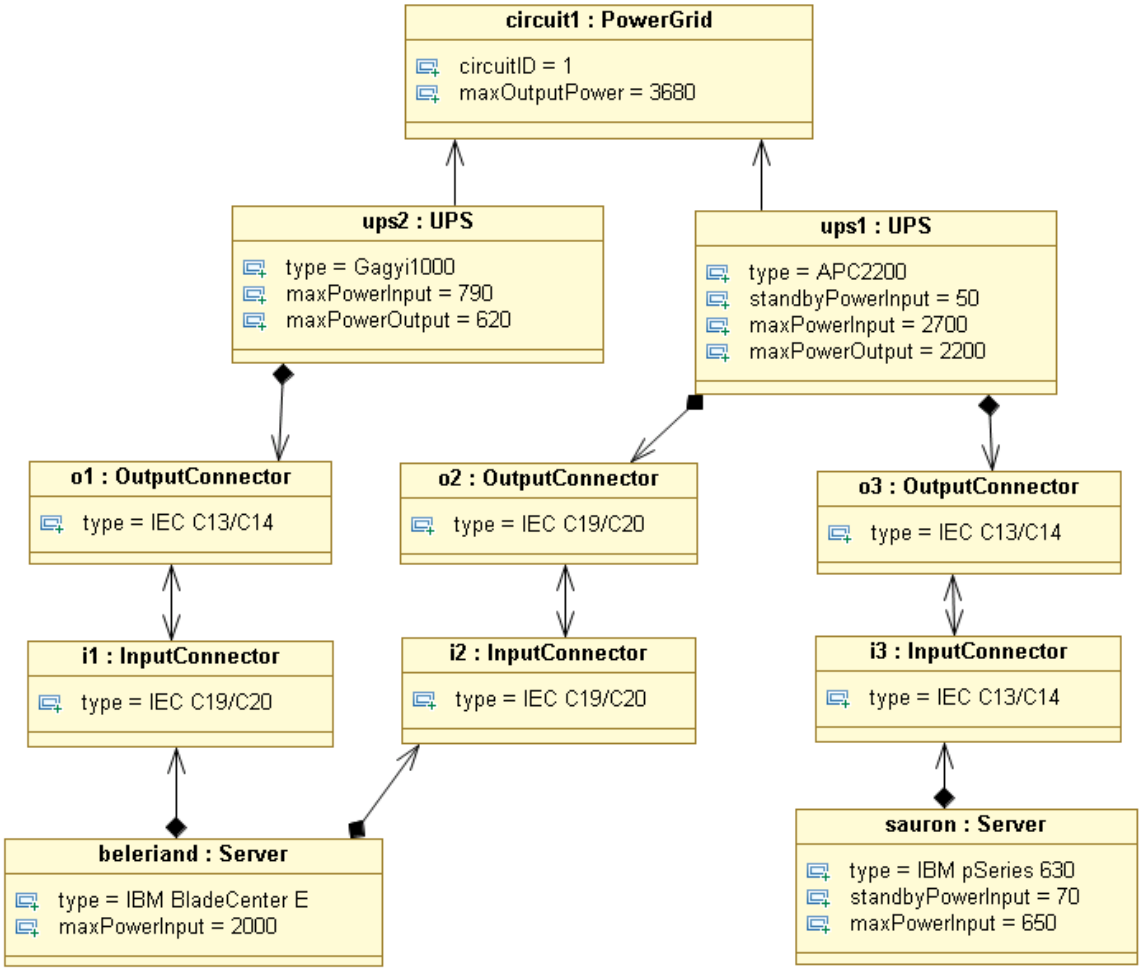
Készítse el egy olyan UML objektumdiagram példányát ennek, ami modellez egy internetszolgáltatót, melynek legalább két kiszolgáló szervere van, melyek közül az egyik DNS, a másik DNS és Web szolgáltatást is nyújt. A DNS szerverek minimum 1 bejegyzést, a web szerverek minimum 2 weboldalt szolgálnak ki.

3.8 Tápellátás

A szerverszobát behálózó vezetékek közül a gépek tápellátását biztosító erősáramú kábelek bekötésének dokumentálására valamint vizsgálatára a következő metamodellt dolgoztuk ki:



Típushelyes példánya-e a következő modell a fenti metamodellnek? Ha nem, adja meg, hogy milyen hibák találhatóak a modellpéldányban!



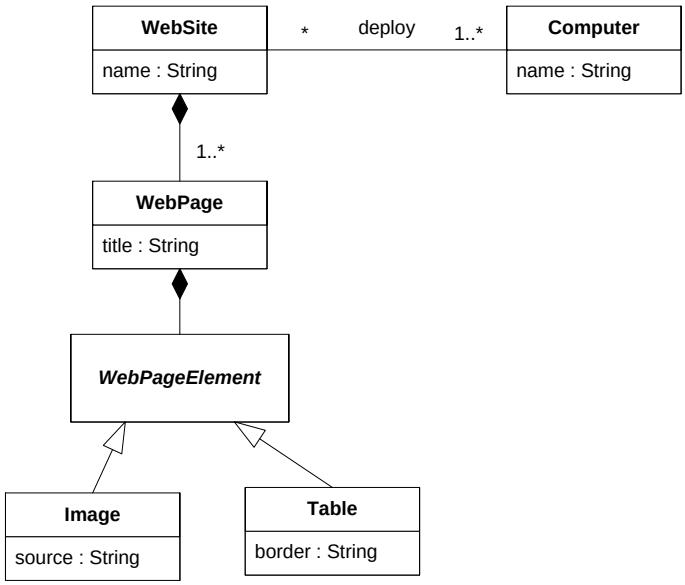
Azon túl, hogy a modell típushelyes példánya legyen a metamodellnek, szeretnénk a bekötés helyességét is vizsgálni. Határozzon meg legalább két – metamodellben nem feltétlenül szerepeltethető – feltételt, amit mindenképpen érdemes lenne ellenőrizni egy erősáramú hálózat üzembiztos működése érdekében!

4 Korábbi vizsgafeladatok

A következő fejezetben korábbi vizsgafeladatokat gyűjtöttünk összes, hasonlókra lehet számítani. Érdemes a frissebbek megoldásával kezdeni a gyakorlást.

4.1 Fürtök bevezetése (2009. 05. 15.)

Adott a következő metamodell, mellyel IT infrastruktúrák egy részletét lehet leírni:



1. Módosítsa úgy a metamodellt, hogy bevezeti a fürt fogalmát! A fürt legalább egy számítógépből áll, és külön neve és IP címe van. (5 pont)
2. Készítsen egy olyan példány modellt, melyben egy két lapból álló webhelyet egy két csomópontból álló fürtre telepítünk! Minden weblap legalább egy elemet tartalmazzon! (15 pont)

4.2 Háttértárak modellezése (2009. 05. 27.)

1. Készítsen egy olyan metamodellt mellyel leírhatóak a háttértárak következő alapfogalmai és a közöttük lehetséges kapcsolatok: merevlemez, partíció, fájlrendszer! Néhány jellemző attribútumot is vegyen fel! A partíciók esetén nem kell figyelembe venni a PC partíciós tábla sajátosságait (4 elsődleges, kiterjesztett, C/H/S címzés), de egyéb általános alaptulajdonságokat (sorszám, bootolhatóság, kezdőcím, méret, típus) jelölje! A metamodell alkalmas legyen többféle fájlrendszer megkülönböztetésére is! (5 pont)
2. Egészítse ki a metamodellt, hogy logikai kötetek leírására is alkalmas legyen! A logikai kötetkezelő képes pillanatkép készítésére is, így az ehhez szükséges alapfogalmak is jelenjenek meg a metamodellben! (7 pont)
3. Készítse el a következő rendszer modelljét az imént kidolgozott metamodell példányaként. Egy gépben egy 36GB-os SCSI és egy 500GB-os SATA merevlemez található. A SCSI merevlemez elején egy 10GB-os rendszerpartíció található, melyen egy RedHat Enterprise Linux foglal helyet. A SCSI lemezen található maradék hely és a teljes 500GB-os merevlemez pedig egy kötetcsoport tagja, melynek neve „StoreVG”. A StoreVG-n definiált 2 darab egyenként 150 GB-os logikai kötet mindegyikét egy-egy Windows Server használja (SAN-on keresztül, aminek nem kell megjelennie a modellben). A logikai kötetek közül az egyikről készült egy pillanatkép 30 GB-os méretben. Feltételezheti, hogy a GB-ot mindenütt egyforma egységekben számolják. (8 pont)

4.3 Hálózati eszközök modellezése (2009. 06. 03.)

a) Készítsen metamodellt hálózati eszközök nyilvántartására. A számítógépeinkben különböző sebességű és gyártójú hálózati kártyák lehetnek. A számítógépek hálózati kábelekkel vannak összekötve, melyekről tárolni szeretnénk, hogy milyen típusúak. Végül az egyes gépek vagy cross-kábel vagy pedig switchek segítségével vannak összekötve. Vannak menedzselhető és nem menedzselhető switcheink is. Végezetül szeretnénk tárolni, hogy mik a számítógépek IP beállításai (IP cím, alhálózati maszk, DHCP használata, stb.). (10 pont)

b) A fenti metamodellnek készítse el egy példány modelljét, ami a következő eszközöket modellezi. A server1 és server2 gépek egy nem menedzselhető switch segítségével vannak összekötve. CAT-5-ös UTP kábeleket és Gigabites Intel hálózati kártyákat használunk. A switch egy Gigabites 24 portos eszköz. C osztályú címeket használunk, a server1 IP címe 10.40.1.1, a server2 pedig a DCHP-től a 10.40.1.10 címet kapta. A server2-ben van ezen kívül van egy 100 Mbites Intel hálózati kártya, amit jelenleg nem használunk. (4 pont)

c) A fenti metamodell nem tartalmaz néhány, a szerverekben használatos megoldást. Módosítsa a metamodell megfelelő részét (rajzolja le újra külön!) úgy, hogy tartalmazza a több portos hálózati kártyák és a NIC teaming fogalmát. Egy több portos kártyán több különböző port van, amik ugyanolyan sebességűek, de külön-külön MAC címmel rendelkeznek. Egy team két hálózati kártya összefogását jelenti, melyek kívülről egy kapcsolatként látszanak (tehát pl. ugyanaz az IP címük). (5 pont)

d) A kiegészített metamodellben nem lehet minden jólformáltsági kényszert pusztán osztálydefiníciókkal és multiplicitásokkal megadni. Mondjon egy ilyen kényszert! (1 pont)

4.4 Számítógép felépítésének modellezése (2009. 06. 17.)

1. Számítógépek belső felépítésének modellezéséhez készítsen egy metamodellt! A számítógépben van egy alaplap, melyről szeretnénk tárolni a chipsetének típusát. A számítógép processzoráról a frekvenciáját akarjuk nyilvántartani. A memóriák kapcsán a memóriák méretét és típusát akarjuk feljegyezni, valamint azt, hogy melyik modul melyik memóriafoglalatban van. A gépben SATA vagy SCSI merevlemez-vezérlők lehetnek, az ezekre kötött merevlemezekről azok méretét és fordulatszámát tároljuk. Minden elem kapcsán tároljuk még a gyártót és a termék azonosítóját. (10 pont)
2. Készítsen el egy példánymodellt, ami a következő számítógépet reprezentálja: Intel Core 2 Duo 2.0 GHz-es CPU, 2x2GB Kingston DDR3-800-as RAM (az egyes és hármas memóriafoglalatban), Intel DP45SG alaplap Intel P45 Express chipsettel, és két 500 GB-os Samsung SATA merevlemezzel. (5 pont)
3. Egészítse ki a metamodellt, hogy a következő szerverekben használt funkciókat is meg lehessen benne adni: többprocesszoros gépek, ECC-t (error checking code) támogató memóriamodulok használata, memory mirroring (azaz megadható, hogy az egyik memóriamodul tartalmát egy másikba tükrözze az alaplap). (5 pont)

4.5 Címtárak modellezése (2010. 06. 14.)

Egy tanácsadó cég különböző vállalatok számra központi címtárak kiépítésével foglalkozik. A kiépítés folyamatának automatizálásához egy megfelelő metamodell tervezését tűzték ki célul, melynek segítségével a megrendelőik igényeit precízen is le tudják majd írni.

A céges infrastruktúrák lényeges elemei az azt alkotó számítógépek és a közöttük futó hálózati kapcsolatok. Ezek a kapcsolatok lehetnek közvetlen, vagy valamilyen hálózati eszközön, switchen vagy hubon keresztül történő összeköttetések. A switch esetében tárolni szükséges a menedzsment IP címét. Minden számítógépen különböző szolgáltatások futhatnak, melyeket authentikáció szempontjából a cég szerver és kliens osztályokba csoportosít. A szerver jellegű szolgáltatások közös jellemzője, hogy melyik

hálózati interfészen, milyen porton várják a kliensek csatlakozását. Ebbe a csoportba tartozik az Active Directory szolgáltatás, amit a tartomány teljes neve jellemez és az OpenLDAP szolgáltatás, amit a RootDN attribútuma ír le. A kliens csoportba tartozó szolgáltatások mindegyike kapcsolódik legalább egy központi címtár szolgáltatáshoz.

a) Készítse el a metamodellt hogy a fent megfogalmazott modellezési céloknak eleget tegyen! (6 pont)

b) A cég egyik ügyfele vegyesen Windows és Linux kliensekkel is dolgozik, melyeken különböző szolgáltatások futnak. A megrendelés egy Active Directory és egy LDAP címtár szolgáltatást is tartalmaz. Az előbbin alapulva működik a Windows fájlmegosztás authentikációja, a webszerver pedig az LDAP szolgáltatást használja fel. Az Active Directory és a windowsos fájlszerver külön gépen fut, a webszerver és az LDAP szolgáltás pedig egy harmadik, linuxos gépre lett telepítve. Készítsük el a konkrét megrendeléshez kapcsolódó példánymodellt a korábban megtervezett metamodell alapján! Jelenítse meg azokat az attribútumokat is, amiknek az értékét nem adtuk meg. (4 pont)

4.6 Open Compute szerverek (2011. 06. 01.)

A Facebook néhány hónappal ez előtt az Open Compute Project nyílt forrású projekt keretében publikálta az oregoni Prineville-ben kialakított adatközpontjuk specifikációit. A fejlesztésnél a legfontosabb célként az energiahatékonyságot tartották szem előtt, és ennek megfelelően alakítottak minden részegységet az optimum eléréséhez. A sok egyedi fejlesztés mellett természetesen itt is nagyon hasonló a felépítés más adatközpontokhoz, azonban a meglévő modellekkel nem lehetne kényelmesen leírni a konfigurációt. Ezen probléma leküzdésére a jelen feladat a konfiguráció leírását lehetővé tevő metamodell kialakítása.

A rendszer alapegysége a szerver[4]. A szervereket modellek alapján állítják össze. A szerver modell leírja, hogy az milyen lemezeket, milyen villamos tápegységeket, ventillátorokat és alaplapot tartalmaz. Az alaplapon találhatóak a CPU-k (a típussal azonosítjuk), I/O portok (lehet SATAII, Ethernet és USB) és memória (a típussal azonosítjuk). A szervereket három oszlopos szekrényekben (triplet) tárolják, minden oszlopban 30-at. Az esetleges áramingadozások és áramszünetek átvészelésére szünetmentes tápegységeket használnak, amelyek egyenként két triplet áramellátására képesek. A teljes adatközpont a párba állított tripletekből és az áramellátásukat biztosító szünetmentes tápegységekből épül fel.

a) Készítsen el egy metamodellt, amely a fent leírt konfiguráció leírását lehetővé teszi! (4 pont)
b) A Facebooknál kétféle szervermodellt használnak: az AMD és az Intel alapút. A szerver modellek adott paraméterekkel rendelkeznek (CPU, memória…), és ezek alapján állítják össze a szervereket. Az egyes szerverek alkatrészei legfeljebb a gyári számokban különböznek egymástól. Készítsen egy példány modellt a fenti metamodellhez amely leírja a következő képzeletbeli AMD szervermodellt.
FBAMD-001: Open Compute Project AMF alaplap (két AMD Opteron 6100 sorozatú processzor hellyel, 24 memória foglalattal, 6 SATAII porttal, 3 USB porttal), két Opteron 6132 HE processzor, 2 * 16GB RDIMM memória (DDR3 PC12800 ECC), 2 darab HD204UI típusú merevlemez. A merevlemezek a 0-s és 1-es SATA portra csatlakoznak. Az áramellátást egy Open Compute Project 450W tápegységgel biztosítják. (4 pont)
c) A fenti példányhoz tartozó információk egy részét nem biztos, hogy meg lehet adni az a) feladatban megadott metamodellben. Mivel kéne még azt kiegészíteni, hogy minden fontos információ bekerülhessen? (2 pont)

4.7 Biztonsági mentések modellezése (2011. 06. 15.)

a) Biztonsági mentést tervezünk az infrastruktúránkhoz, melyhez az automatizálást elősegítendő egy metamodell készítése a feladatunk. Első körben „pull” típusú mentést szeretnénk végrehajtani,

[4] A projekt dokumentációjában a szervereket nem említik, csak a bezáró egységként szolgáló keretre (chassis) hivatkoznak. Az egyértelműség kedvéért a feladat szövegében eltérünk ettől a konvenciótól.

melyhez a megfelelő eszközt már ki is választottuk. Az eszköz alapeleme a biztonsági mentés kötet, melyhez tartoznak a különböző kliensek. A kötethez tároljuk a fájlrendszerbeli elérési útját, míg a különböző kliensekhez azok nevét, IP címét és azt az időinformációt, hogy mentést mikor kell készíteni róla (ez jellemzően egy napon belüli időpont). Minden klienshez tárolunk pillanatképeket, melyeknek rögzítjük a készítés dátumát és méretét. A pillanatképek lehetnek teljes mentésből származóak vagy inkrementális jellegűek. Ez utóbbi esetén tároljuk azt, hogy melyik korábbi pillanatképhez képest készültek. A mentéseket nem végezzük el minden esetben a kliens teljes fájlrendszerén, gyakran alkalmazunk tiltó listákat, melyek azon könyvtárak elérési útjait tartalmazzák, amik kimaradnak a mentésből. Tiltó listát lehet globálisan megadni a mentési kötethez, vagy pedig lehet kliensenként definiálni. (4 pont)

b) A fenti metamodellhez készítsen el egy példánymodellt. A `/backupvolume` útvonalon elérhető mentési kötethez 3 kliens, a *yoda* szerver (192.168.5.23), a *luke* szerver (192.168.5.24) és a *vader* szerver (192.168.5.13) tartozik. Globálisan tiltjuk a `/proc`, `/sys` és `/dev` könyvtárak mentését, míg a *luke* szerveren a `/media/images` lokálisan is tiltva van. A kliensek mentése sorban 1:00, 2:00 és 3:00 időpontokban fut le minden nap hajnalban. A mentést ezen a héten indítottuk el, így eddig egy teljes mentés készült minden kliensről 2011. 06. 13-án és egy erre épülő inkrementális a tegnapi napon. (4 pont)

c) A biztonsági mentésért felelős rendszerünket a továbbiakban ki akarjuk egészíteni windowsos hosztokhoz is használható „push” típusú mentés lehetőségével. Gondoljuk át, hogyan kéne kiegészíteni a metamodellt, hogy az ilyen jellegű mentéseket is támogassa! A teljes metamodellt nem kötelező újra lerajzolni, de egyértelműen jelöljük a kiegészítéseket. (2 pont)

4.8 BackBlaze pod fizikai modell (2012.05.30.)

A *BackBlaze* cég egy néhány éve indult, online backup szolgáltatást nyújtó kisvállalat. A költséghatékonyság jegyében sajáttervezésű szerverekkel oldják meg az adattárolást, ezeket *pod*oknak nevezik. Egy pod lényegében egy PC alapokra épülő vezérlő számítógép és néhánytucat merevlemez, egy 4U magas házba helyezve.

a) A dinamikusan fejlődő cég iránt több befektető is érdeklődik, ezért a nyilvántartások terén eddig uralkodó káoszt fel kell számolni és minden fontosabb alkatrész sorozatszámát és néhány adatát el kell tárolniuk.[5] Készítsen metamodellt az alább leírt követelményeknek megfelelően a szükséges adatok tárolására! (5 pont)

A podok rackekben állnak, egy rackben maximum 10 pod lehet. Az idők folyamán két típusú podot készítettek, ezeket frappánsan Pod 1.0-nak és Pod 2.0-nak hívják. A Pod 1.0-ban 3 db PCIe és 1 db PCI port áll rendelkezésre SATA-vezérlők fogadására, míg a Pod 2.0-ban csak 3 db PCIe port. A Pod 1.0-ban kétportos PCIe SATA-vezérlőket és négyportos PCI SATA-vezérlőt használnak, míg a Pod 2.0-ban minden használt (PCIe) SATA-vezérlő négyportos. Minden SATA-vezérlő minden portjához csatlakozhat egy merevlemez vagy egy SATA-elosztó, amely 5 merevlemez csatolását teszi lehetővé.

Sorozatszámmal rendelkezik minden rack, pod, SATA-vezérlő és merevlemez. Tároljuk továbbá a podokban lévő processzor típusát és a memória méretét, illetve a SATA-vezérlők márkáját, valamint a merevlemezek márkáját és kapacitását.

(*Egyértelműsítő megjegyzések:* SATA-elosztóhoz további SATA-elosztó nem csatlakozhat. PCI porthoz csak PCI SATA-vezérlő, PCIe porthoz csak PCIe SATA-vezérlő csatlakozhat. A podokban természetesen található alaplap, processzor, tápegység stb., azonban ezeket nem szükséges modellezni. A SATA-elosztók nem rendelkeznek sorozatszámmal. A modellnek nem szükséges teljesen általánosnak lennie, elegendő a fenti specifikációnak megfelelnie.)

[5] A feladat többnyire valós adatokon alapul (lásd bővebben http://blog.backblaze.com/category/storage-pod/ – természetesen csak a vizsga után), kivéve a káoszt: természetesen nem feltételezzük, hogy nincs megfelelő eszköznyilvántartásuk.

b) Készítsen az a) feladatban megtervezett metamodelljéhez egy példánymodellt az alábbi adatok alapján!

Az R001 sorozatszámú rackben egyetlen P001 sorozatszámú Pod 1.0 árválkodik, amelyben Intel E8600 processzor és 4 GB memória található. Ebben mind a négy lehetséges bővítőhelyen található 1-1 megfelelő SATA-vezérlő (sorozatszámuk: SC001..SC004, a PCI interfészű vezérlő márkája Addonics, a többié Syba). Mindhárom PCIe SATA-vezérlőhöz 2-2, a PCI SATA-vezérlőhöz 3 darab SATA-elosztó csatlakozik. A SATA-elosztókhoz összesen 45 darab merevlemez csatlakozik, sorozatszámuk HD01..HD45, mindegyik Seagate gyártmányú, 1,5 TB-os. (4 pont)

(Természetesen nem szükséges az összes SATA-elosztót és merevlemezt felrajzolnia, elegendő mindegyikből 1–2-t, amelyik jól reprezentálja a kapcsolatait.)

c) Milyen feltételeket (kényszereket) nem tudott kifejezni megfelelően a metamodellben? Adjon meg legalább egyet. (1 pont)

4.9 BackBlaze pod logikai modellje (2012.05.30.)

A *BackBlaze* cég egy néhány éve indult, online backup szolgáltatást nyújtó kisvállalat. A költséghatékonyság jegyében sajáttervezésű szerverekkel oldják meg az adattárolást, ezeket *pod*oknak nevezik. Egy pod lényegében egy PC alapokra épülő vezérlő számítógép és 45 darab merevlemez, egy 4U magas házba helyezve.

A podok rackekben állnak, egy rackben maximum 10 pod lehet. Egy-egy podban a 45 darab merevlemez 3, egyenként 15-15 lemezből álló RAID6 tömbbe van rendezve.[6] Minden egyes felhasználó adatai pontosan egy RAID6 tömbön tárolódnak. Karbantartást, hibás lemezek cseréjét azonban csak leállított podon végeznek, ilyenkor nyilván az összes olyan kötet elérhetetlen lesz, amelyet az adott pod szolgál ki. Minden podhoz és minden rackhez ki van jelölve 1-1 felelős karbantartó. Egy pod leállítása csak a felelős karbantartójának jelenlétében tehető meg. Bizonyos esetekben szükséges a rackhez tartozó felelős karbantartó jelenléte is. Egy karbantartó személyhez maximum 10 eszköz (rack vagy pod) tartozhat.

Minden RAID tömbről tároljuk továbbá a tárolható adatmennyiséget, a telítettség mértékét (%-ban), illetve az üzemképes merevlemezek számát.

(*Egyértelműsítő megjegyzések:* A RAID-tömböknél alacsonyabb szintet nem kell modellezni.)

a) A podok 1.0 verziója esetén 1,5 TB-os merevlemezeket használnak adattárolásra. Hány hibát képes tolerálni 1-1 15 lemezes RAID6 kötet és mekkora adatmennyiség tárolható rajta? (1 pont)

b) Készítsen metamodellt, amely segítségével tárolhatók a felhasználók és a hozzájuk rendelt kötetek. Az elkészítendő metamodellből továbbá kiderül, hogy egy rack vagy egy pod karbantartása milyen kiesést okozhat, illetve hogy ki felelős az adott karbantartási feladatért, azaz az ilyen jellegű, fent specifikált adatokat is tárolni kell tudni. (4 pont)

c) Készítsen példánymodellt az előzőekben elkészített metamodellhez az alábbiak alapján! (4 pont)

Egyelőre egyetlen rackünk és benne két podunk van (P1, P2). A P1 podon található három RAID-tömb R1, R2 és R3. Jane és Jack felhasználó adatai jelenleg az R1, John adatai pedig az R2 köteten tárolódnak. Az R1 kötet 14, a többi 15 üzemképes merevlemezt tartalmaz. Mindegyik tömb 1,5 TB-os merevlemezekből áll, mindegyik kötet telítettsége 30%. Tudjuk továbbá, hogy az egyetlen rack és a P1 pod karbantartója Rob, a P2 pod karbantartója Rebeca.

d) Hogyan kéne átalakítania az elkészített metamodelljét, ha vegyesen használnának RAID5 és RAID6 tömböket is? (1 pont)

[6] A feladat eddigi része valós adatokon alapul (lásd http://blog.backblaze.com/category/storage-pod/ oldalt bővebb információkért – természetesen csak a vizsga után), innentől bizonyos elemei a feladatíró képzeletének szüleménye.

4.10 Infrastruktúra konfigurációjának modellezése (2012.06.06.)

a) Vállalatunk vezetése elhatározta, hogy költséghatékonysági szempontból az infrastruktúra menedzsmentje és dokumentálása során ugyanazt a modellezési keretrendszer használjuk. Több különböző rendszer megvizsgálása után egy, a saját igényeinket kielégítő metamodell elkészítése mellett döntöttünk, alapján automatikusan elkészíthető az infrastruktúra konfigurációja és dokumentációs célokra is megfelelő. Infrastruktúránkban különböző szerverek és közöttük lévő függőségeket szeretnénk ábrázolni. A szerverekről nyilvántartjuk a processzor, a memória és a hálózati paramétereiket, valamint azt, hogy virtuális vagy fizikai szerverekről van szó. Hálózati paramétere minden szervernek legalább egy, de akár több is lehet. Minden virtuális szerver esetén fontos, hogy melyik fizikai szerveren foglal helyet, míg a fizikai szerverek esetén a polc száma kerül tárolásra, ahol elhelyezkedik. A szerverek közötti függőségek két típusát különböztetjük meg: a szigorú függőség esetén az adott szerver a függőség meg nem léte esetén el sem tud indulni, míg a laza függőség nem akadályozza a boot folyamatot, de bizonyos szolgáltatások nem fognak működni a rendszeren. A függőségek esetén fontos tárolni azok szöveges leírását és meg kell különböztetni a függőség irányát is. (4 pont)

b) A fenti metamodellhez készítsen el egy példánymodellt. Az `Anakin` szerver kettő AMD Optheron processzorral, 4 GB memóriával rendelkező fizikai szerver a 3. polcon, míg a `Luke` és `Leia` szerverek egy-egy virtuális gépek az `Anakin` szerveren és egy-egy dedikált AMD Optheron processzorral és 1-1 GB memóriával rendelkeznek. A fizikai gép két hálózati interfésszel rendelkezik, míg a virtuális gépek egy-egy csatolóval. A `Luke` szerver biztosítja a `Leia` számára a fájlmegosztást, ami a `Leia` webszervere által kiszolgált fájlokat tartalmazza. (4 pont)

c) Az infrastruktúra modellben később szeretnénk tárolni a szolgáltatásokat, amiket a szervereink nyújtanak. A függőségeket ennek megfelelően finomítani kell úgy, hogy szolgáltatások között értelmezett függőségek is legyenek a rendszerben. Gondoljuk át, hogyan kéne kiegészíteni a metamodellt, hogy az ilyen jellegű információkat is támogassa! A teljes metamodellt nem kötelező újra lerajzolni, de egyértelműen jelöljük a kiegészítéseket. (2 pont)

4.11 Szoftverfejlesztési projektek és eszközök (2012.06.13.)

A szoftverfejlesztés támogatása céljából vállalatunk új eszközöket szeretne bevezetni. Tanulva a korábbi konfigurációs bonyodalmakból és jogosultságkezelési problémákból most tervezetten, előre átgondoltan és folyamatosan karbantartható módon szeretné a bevezetést meglépni.

A vállalat összetett szoftverrendszereket fejleszt partnerei számára. Minden szoftverrendszer több modulból épül fel, melyeket önálló fejlesztőcsoportok fejlesztenek egymástól teljesen függetlenül. Minden szoftvermodulhoz biztosít a vállalat önálló SVN repository-t és TRAC felületet. A hozzáférési jogosultságokat olvasás és írás szerint, de a teljes SVN és TRAC szintjén lehet állítani, finomabb beállítást nem engedélyeznek. Három különböző felhasználói csoportot különböztetünk meg: vannak partnerek, akik a szoftverrendszer megrendelői, vannak vezető fejlesztők, akik a modulok integrációjáért felelősek és vannak fejlesztők, akik egy-egy modul fejlesztésén dolgoznak. Jogot mindig egyes felhasználók kaphatnak, és alapértelmezésben az adott modult fejlesztők és a vezető fejlesztő írási joggal rendelkezik, míg a partnerek olvasási joggal bírnak.

a) Feladatunk, hogy elkészítsünk egy megfelelő metamodellt ami alapján a konfiguráció és jogosultságkezeléshez szükséges példány modellek megalkothatóak és azok alapján a technológiai beállítások automatikusan elvégezhetőek. (5 pont)

b) A fenti metamodellhez készítsen el egy példánymodellt. A `Fluxuskondenzátor` szoftverrendszert a Sötét Erők nevében `Darth Vader` nagyúr rendelte meg a fejlesztő csapatunktól. A rendszer két modulból épül fel, melyek integrációjáért `János`, a vezető fejlesztő felel. A `Fluxus` modulon ketten `Aladár` és `Béla` dolgoznak, míg a `Kondenzátor` modulon egyedül `Dániel` munkálkodik. (3 pont)

c) A fejlesztések során a vállalat rájött, hogy a rendszer nem elég rugalmas, mert újabb és újabb szoftverfejlesztést támogató rendszerek bevezetésével minden alkalommal minden érintett felhasználónak jogot kell adni. Szeretnék bevezetni a felhasználói csoport fogalmát a rendszerbe és a csoportba tartozás szerint kiosztani a jogokat.(2 pont)

4.12 CERN Advanced Storage Manager (2013.06.05.)

A CERN Nagy Hadron Ütköztetője (LHC) rengeteg adatot termel (~10 PB évente), amelyek tárolása komoly problémákat jelent. A CERN kutatói kifejlesztették a CERN Advanced Storage Manager-t (CASTOR), hogy a problémán úrrá tudjanak lenni. A rendszer alapvetően diszkes tárolást alkalmaz, azonban a régebbi adatokat szalagos tárakra helyezik át. A CASTOR adatközpontokban nagyteljesítményű diszkszerverek vannak, minden szerverben több, nagykapacitású merevlemezzel. Az adatközpontok másik igen fontos eleme, az archiváló rendszer, amely olyan szervereket tartalmaz, amelyek mindegyike több szalagos tárral rendelkezik.

Amikor egy kliens egy kérést intéz a rendszer felé, akkor az adatközpont kéréskiszolgáló szervere fogadja, amely létrehoz egy kliensfeladatot. Az adatközpont globális ütemezője fogja kezelni a kliensek kiszolgálására létrehozott feladatokat (ez egy teljesen önállóan futó komponense az adatközpontnak), valamint az archiválószerver által létrehozott archiváló feladatokat is. Egy klienstől érkező kérés kiszolgálásához a rendszer a megfelelő diszkszerverhez fordul. Ha a diszkszerverről az archiváló már archiválta az adatokat, akkor a diszkszerver a kérést továbbítja az archiválórendszer felé. Az archiválórendszer archiválási feladatait a lemezek elemzése után, egy belső algoritmus alapján hozza létre, amely során egy teljes diszk tartalmát átadja egy szalagos tárolót tartalmazó szervernek.

a) A feladatunk, hogy egy UML osztálydiagram segítségével szemléltessük a rendszer elemeit és azok kapcsolatait! Gondoljuk végig, hogy az egyes osztályoknak milyen tulajdonságokat szükséges nyilvántartaniuk, és ezek közül legalább kettőt jelenítsünk meg minden osztály esetén! (7 pont)

b) A fenti modellnek készítsük el egy olyan példányát, ami egy olyan adatközpontot ábrázol, ahol két diszkszerver (két-két lemezzel) és egy archiváló rendszer van (egy szalagos tárral). Az adatközpont egyetlen kéréskiszolgálójához jelenleg két kliens csatlakozik, az egyik épp most küldött egy kérést. (3 pont)

4.13 Incidenskezelő rendszer (2013.06.05.)

Egy nagyméretű informatikai infrastruktúrában fontos, hogy minden hibát mielőbb kezelni tudjunk, amihez a megfelelő incidenskezelő rendszer elengedhetetlen. Egy ilyen rendszer kifejlesztésével bíztak meg minket, amelyhez egy UML osztálydiagram elkészítése most a feladatunk.

Az incidenskezelő alapgondolata az, hogy egy függőségi gráfot tárol az informatikai infrastruktúráról. Alapvetően kétféle kapcsolat lehet két alkotóelem között: függés (amikor logikailag függünk egy másik komponenstől) illetve tartalmazás (amikor fizikailag is függünk tőle, ha a tartalmazott komponens kiesik, akkor a másik működése azonnal leáll). Az incidenskezelő pillanatnyilag csak a szervereket, processzorokat, hálózati kártyákat és routereket kezel és más hardverelemekkel nem foglalkozik. Az incidenskezelő bejegyzései (az incidensek) azt tárolják, hogy egy meghibásodás esetén mi az a legalacsonyabb elem, ami a meghibásodást okozza, illetve, hogy milyen kapcsolatokon keresztül milyen további elemek esnek ki. (Itt fontos az, hogy pontosan tudjuk, hogy melyik kapcsolaton melyik elem esett ki, tehát a kettő közti egyértelmű összerendelés!)

a) A feladatunk, hogy egy UML osztálydiagram segítségével szemléltessük a rendszer elemeit és azok kapcsolatait! Gondoljuk végig, hogy az egyes osztályoknak milyen tulajdonságokat szükséges nyilvántartaniuk, és ezek közül legalább kettőt jelenítsünk meg minden osztály esetén! (7 pont)

b) A fenti modellnek készítsük el egy olyan példányát, ahol egy szerver és egy router szerepel, mindkettőben természetesen van processzor és hálózati kártya. A szerver a routeren keresztül

kapcsolódik az internetre, és egy publikus honlapot szolgál ki. Az incidenskezelőben két incidens szerepel: a router hálózati kártyája meghibásodott, illetve a szerver processzora kiesett. (3 pont)

4.14 Hadoop-alapú adatfeldolgozó rendszer (2013.06.12.)

Az országon átvonuló árvíz nem csak károkat és feladatokat jelent, hanem kiemelkedően fontos statisztikai adatokat is biztosít a vízügyi kutatók és az operatív munkában résztvevők számára. Az árvízzel kapcsolatos különböző adatokat az országban elhelyezett több ezer szenzor rögzíti és juttatja el az operatív törzs által üzemeltetett adatgyűjtő központba, ahol statisztikai elemzéseket és előrejelzéseket készítenek belőle a későbbi szakaszokon várható vízügyi események szempontjából. Az adatok mennyisége és a délen még folyó védekezés gyors elősegítése miatt az adatok feldolgozására nagyteljesítményű, elosztott, Hadoop-alapú adatfeldolgozó rendszert állítottak üzembe.

A rendszer megbízható működése mellett az üzemeltetők az újkori szellemnek megfelelően a média és közvélemény tájékoztatására is fontos hangsúlyt fektetnek, ezért megkértek minket, hogy a rendszerükhöz készítsünk egy grafikus ábrázolási lehetőséget.

a) Készítsen metamodellt az alább leírt követelményeknek megfelelően a szükséges adatok tárolására!

A rendszer különböző szerverekből áll, amelyek mindegyikéhez tároljuk a processzor órajelét, a magok számát és a memória méretét. Ezen felül 4 különböző típus tudunk megkülönböztetni:

- DataNode: egyszerű elosztott adattárolásra alkalmas és tároljuk az összesített tárkapacitását.
- NameNode: metaadat tárolását végzi el, ahol tárolni kell a fájlrendszeri bejegyzések számát.
- JobTracker: a futó adatbányászati folyamatokat (Job) tartja nyilván, ahol tárolni kell a maximálisan megengedett párhuzamosan futó folyamatok mennyiségét
- TaskTracker: ténylegesen futtatja a folyamatok részfeladatait (Task), ahol az aktuálisan futó feladatok számát és az felhasznált memória mennyiségét tároljuk.

Minden DataNode-hoz tartozik legalább egy, maximum 8 merevlemez, amihez tároljuk a gyártó és a típus jellemzőket. A szerverek hálózati kapcsolaton keresztül érik el egymást. Minden szerver csatlakozik egy switch-hez, amiről tároljuk a típusát és portjainak számát. Szerver csak switch-hez csatlakozhat, de két switch közvetlenül össze lehet kötve egymással.

A JobTracker szervereken regisztrált adatbányászati folyamatokhoz (Job) tároljuk a nevüket és a beküldő felhasználó nevét. A TaskTrackeren futó részfeladatokról (Task) nyilvántartjuk a méretüket és azt, hogy mekkora részei az eredeti feladatnak. Minden részfeladat egyértelműen kapcsolódik egy adatbányászati folyamathoz és egy adatbányászati folyamat legalább egy részfeladatot tartalmaz.

b) Készítsen az a) feladatban megtervezett metamodelljéhez egy példánymodellt az alábbi adatok alapján! A rendszerben 1-1 NameNode és JobTracker és 2-2 DataNode és TaskTracker üzemel, amik elosztva, összesen két, egymáshoz kapcsolódó switch-en keresztül kommunikálnak egymással. A rendszerben két adatbányászati folyamat fut (banyasz1 és banyasz2 néven), amik 2-2 részfeladatra tagolódnak.

c) Milyen feltételeket (kényszereket) nem tudott kifejezni megfelelően a metamodellben? Adjon meg legalább egyet.

4.15 Adatközpont monitorozó rendszere (2013.06.19.)

Adatközpontunkban egy olyan monitorozó rendszert alakítunk ki, amely részben saját és részben gyári komponenseket is tartalmaz. A gépteremben 2 sornyi rackszekrény található, mindegyik sorban 3-3 szekrény (most épült még csak meg a gépterem nemrég). Minden rackszekrény elejére és hátuljára elhelyeztünk egy-egy saját gyártású hőmérsékletszenzort, a sorok közepére egy-egy pára- és tűzjelző szenzort. A szenzorok Ethernet hálózaton keresztül csatlakoznak az adott sorban elhelyezett kis beágyazott eseménygyűjtő komponenshez. Az egyes sorok eseménygyűjtői fogadják a hozzájuk tartozó

szenzorok adatait, egybegyűjtenek egy 5 perces intervallumot, majd azt továbbküldik a központi monitorozó rendszerhez (ez egy WS-Management felületet biztosító, webes felülettel rendelkező alkalmazás). A központi monitorozó rendszerbe ezen kívül be lehet regisztrálni többféle szabványos protokollon keresztül lekérdezhető komponenst, amiket így figyelni tud. Jelenleg az SNMP és WS-Management protokollokat támogatjuk. A központi rendszerbe már bekötöttük az egyes rackszekrényekben található menedzselhető kapcsolókat (switch), amiket SNMP-n keresztül érünk el.

A felügyeleti rendszerhez kell egy vizualizációs komponenst is készíteni. Mivel a gépterem elrendezés folyamatosan bővül, így nem akarjuk fixen belerakni a jelenlegi állapotot, hanem készítünk egy általános, az adott környezetet leíró modellt, majd ennek egy-egy példányát jelenítjük meg dinamikusan.

a) Készítsük el a modellt, és ábrázoljuk UML osztálydiagram formájában (ne felejtsük el a számosságokat megadni, ahol tudjuk). A modell tartalmazza a fizikai és a logikai elrendezésre vonatkozó információkat is! (6 pont)

b) Adjuk meg a jelenlegi géptermi elrendezést egy típushelyes példánymodell formájában (elég csak az egyik rackszekrény sort ábrázolni). (3 pont)

c) Közben kiderült, hogy a szenzorok és az eseménygyűjtők kapcsolatára jobb lenne esetleg más protokollokat használni (pl. ZigBee, Wi-Fi). Módosítsuk a modellt, hogy ez az információ is bekerüljön. (1 pont)

4.16 MMORGP infrastruktúra (2014.06.02.)

Manapság igen népszerűek az MMORPG játékok, cégünk is ezzel foglalkozik. A játék fejlesztése mellett azonban szükségünk van egy ezt támogató, könnyen skálázható infrastruktúrára is.

A játékosok (Player) egy külön, erre a célra kijelölt szerveren keresztül tudnak bejelentkezni (LoginServer) a játékba. Ahhoz, hogy a játék állapotát nyilvántartó szerver (WorldServer) ne legyen túlterhelt, a játékosok csak proxy-khoz (ProxyServer) csatlakozhatnak. Így lehet megvalósítani a terheléselosztást, valamint ezek a szerverek végzik el az általános hálózati feladatokat (pl. titkosítás, tömörítés). Magáért a játék szimulálásáért a játék szerverek (GameServer) a felelősek. A proxy szerverek és a nyilvántartó szerver az összes GameServer-rel össze van kötve. A játékhoz kapcsoló adatokat egy elosztott adatbázisban tároljuk: ezt úgy kell elképzelni, hogy célszerűen több adatbázis-szerver funkciót megvalósító csomópont (Node) van, azonban ezek közül bármelyikhez fordulunk, megkapjuk a kért adatot. Az adatbázis-szervert csak a bejelentkezést és a világot nyilvántartó szerverekkel kívánjuk összekapcsolni, viszont ezek az összes csomóponttal kapcsolatban vannak.

A rendszer építésekor igyekszünk a hálózati réteget úgy megvalósítani, hogy ez ne jelentsen szűk keresztmetszetet. A hálózati elemek (kábel, switch, router…) modellezése nem feladat, jelenleg a logikai kapcsolatok modellezése a fontos.

a) Készítsen egy UML metamodellt, ami a fent leírt rendszer leírását teszi lehetővé! (4 pont)

b) A tesztrendszerben két proxy szerver, egy játékszerver és egy adatbázis csomópont van. Ehhez összesen 4 felhasználó csatlakozik, egyenlően elosztva. Készítsen erről egy példánymodellt! Ne feledkezzen meg az explicite nem felsorolt, de szükséges elemekről sem! (4 pont)

c) A játékunk befutott a kontinensen, azonban szeretnénk tovább terjeszkedni. Ahhoz, hogy a megfelelő felhasználói élményt nyújtsuk, alacsony válaszidőre van szükségünk. Ehhez a tesztrendszerhez hasonló rendszereket a világ több pontján kell elhelyezni, azonban biztosítani kell, hogy egy felhasználó bármelyik WorldServer-hez tartozó játékban részt vehessen. Hogyan módosítaná az elkészített metamodellt? (2 pont)

4.17 Képfeltöltő szolgáltatás (2014.06.10)

Szeretnénk egy képfeltöltő és megosztó szolgáltatással betörni a piacra nyáron. A fejlesztés során szeretnénk minél jobban kihasználni a modellezés nyújtotta lehetőségeket, ezért már a fejlesztési folyamat elején az architektúrát is modell-alapon kívánjuk megtervezni. A mi feladatunk, hogy elkészítsünk egy metamodellt, amely segítségével majd a konkrét rendszer modelljét le lehet írni. Az architektúra alapjául az egyik igen népszerű szolgáltatás, a Flickr szolgál. Az időszakos nagy terhelések elkerülése érdekében az egész rendszer terheléselosztókon érhető el kívülről. Minden ilyen terheléselosztónak van egy IP címe, illetve pontosan egy másik terheléselosztóval áll kapcsolatban, akihez túlterhelés esetén továbbítja a kéréseket. A terheléselosztók mögött tetszőleges számú szerver állhat. Minden szervernek van egy IP címe és egy neve (ez egy szöveges azonosító, amivel könnyebben tudunk rájuk hivatkozni). Kétféle szervert különböztetünk meg: cache- és alkalmazásszervert. A cache szerverekben legalább egy, és legfeljebb 5 diszk található. A diszkekről tároljuk a méretüket (gigabájtban), a típusukat (HDD vagy SSD) és azt, hogy éppen hány kép van rájuk feltöltve. Az alkalmazásszerver legalább egy cache szervert ismer. A felhasználók adatait és hogy melyik kép kihez tartozik, adatbázisokban tároljuk elosztott módon. Minden adatbázisról tárolni szeretnénk a gyártóját (MSSQL, Oracle, stb...) és a verzióját. Kétféle adatbázis van, a master és a slave adatbázis. A slave adatbázisban tárolódnak ténylegesen az információk elosztott módon. Ezekről az aktuális méretüket tároljuk. A master adatbázisok célja, hogy az alkalmazásszerverek elől elfedjék az elosztottságot úgy, hogy az alájuk rendelt (legalább 2 és legfeljebb 4) slave adatbázisokat egy egységes virtuális adatbázissá vonják össze. Az alkalmazásszerverek tetszőleges számú master adatbázissal állhatnak kapcsolatban.

a) Készítsen el egy metamodellt, amely a fent leírt architektúra leírását lehetővé teszi!

b) Az architektúrát első körben a következőképpen képzeltük el. Két terhelésosztó szolgálja ki a rendszert. Az egyik mögött két cache szerver található, a másik mögött két alkalmazásszerver. Az egyik cache szerverben két HDD, a másikban egy SSD található. Az alkalmazásszerverek mindkét cache szervert elérik. Mindkét alkalmazásszerver alatt egy master adatbázis van, az egyik Oracle, a másik MSSQL termék. Az Oracle alatt három további Oracle slave van, az MSSQL alatt pedig két MSSQL slave. Készítse el az architektúra példánymodelljét. Azon attribútumokat, amikre a leírás nem tért ki (IP címek, nevek) valamilyen kitalált (de értelmes) értékkel töltse fel!

c) A cache szerverek nagyobb megbízhatósága és/vagy teljesítménye érdekében szeretnénk a bennük található diszkekből RAID tömböket képezni. Egy RAID tömbről a típusát (0,1,5,6,stb...) tároljuk, és azt, hogy mely diszkek tartoznak hozzá. Egy diszk csak egy RAID tömbben vehet részt, egy RAID tömbben pedig legalább két diszknek kell lennie. Egészítse ki a metamodellt (elég csak a változás által érintett részt lerajzolni újra)!

4.18 OpenFlow (2014.06.16.)

Az OpenFlow egy nyílt szabvány, mely lehetővé teszi különböző hálózati protokollok gyors prototipizálását, futtatását és tesztelését. Az OpenFlow segítségével switchek, útválasztók és egyéb hálózati eszközök programozhatók fel gyorsan és egyszerűen. A szabványt mára már több cég terméke is támogatja. Cégünket azzal bízták meg, hogy egy szoftvert készítsen, mellyel hatékonyan lehet OpenFlow alapú hálózati topológiákat modellezni.

A hálózatban végpontok (Endpoint) és hálózati eszközök (NetworkDevice) vannak. A hálózati eszköz lehet switch (Switch) vagy útválasztó (Router). Egy topológiában (Topology) bármennyi végpont és hálózati eszköz lehet. A hálózati eszközök bármivel összeköthetők, viszont nem szeretnénk a példánymodellek készítőinek megengedni, hogy két végpontot közvetlenül összeköthessen (viszont egy végpont több hálózati eszközzel is összeköthető). A switch és az útválasztó csak a nevében tér el, ugyanazokkal a funkciókkal rendelkeznek. Minden hálózati eszköznek van gyártója (manufacturer) és típusa (type). Egy hálózati eszköz egy szoftveres és hardveres rétegből épül fel (SoftwareLayer, HardwareLayer). Az előbbi tartalmazza a firmware verzióját (firmwareVersion), utóbbi pedig az ún.

flow table-t (FlowTable). A táblázat bejegyzések listájából áll, ezek szolgálnak forgalomirányítási szabályként (Rule). Példa szabályokra:

devicePort	ipSrc	ipDest	tcpSrcPort	tcpDstPort	action	*description*
*	*	10.0.10.6	*	*	port1	*Továbbítás az 1-es porton*
*	*	10.0.10.6	*	22	drop	*SSH portra érkező csomagok eldobása*

(A szabályok valójában több tagot is tartalmazhatnak, ezzel most nem kell foglalkozni.)

a) Készítsen egy UML metamodellt, ami a fent leírt rendszer leírását teszi lehetővé! (4 pont)

b) A tesztrendszerben egy útválasztó van, amelyre a fenti két szabályt programoztuk fel. Az útválasztó három számítógéppel van összekötve. Készítsen erről egy példánymodellt! Ne feledkezzen meg az explicite nem felsorolt, de szükséges elemekről sem! (4 pont)

c) Hogyan módosítaná az elkészített metamodellt, hogy bármit bármivel (azaz két végpontot is) össze lehessen kötni? (2 pont)

4.19 Tartalomkezelő rendszer (2015.05.27.)

Cégünk szeretne egy tartalomkezelő (CMS) rendszert építeni, amely segítségével a rendszer szerkesztői különféle tartalmakat (hírek, képek, videók, stb...) tudnak eljuttatni a felhasználók felé. A fejlesztési folyamat elején az architektúrát is modellek segítségével kívánjuk megtervezni.

A rendszer alapvetően kliens-szerver architektúrájú. Jelenleg mobilos és webes klienseket tervezünk, de később újabbak is lehetnek. Minden kliensről tároljuk a verziószámot, ezen felül pedig a mobilos kliensek esetén a kliens típusát (Android, iOS, Windows) webes kliens esetén pedig a legnépszerűbb böngészőktől (Chrome, Firefox, IE) elvárt minimális verziószámot. A CMS rendszerekben tipikusan kétféle tartalom van: statikus (például képek, videók, szkriptek) és dinamikus (például hírek, bejegyzések). A statikus és dinamikus tartalmat különböző szerverek szolgálják ki, ugyanis a statikus tartalom szerver alatt közvetlenül a diszkek vannak, míg a dinamikus tartalom szerver alatt adatbázisok. Természetesen az adatbázishoz is mindig kapcsolódik egy diszk. A diszkről a kapacitását és az aktuális foglaltságát, az adatbázisról pedig a gyártóját és a verzióját tároljuk. A kétféle (statikus és dinamikus) tartalom általában vegyesen jelenik meg, ezért szükség van egy alkalmazásszerverre, aki összeállítja a tartalmat. Az alkalmazásszerver mindig pontosan egy dinamikus tartalom szerverrel és két statikus tartalom szerverrel áll kapcsolatban. Minden szervernek tudni kell az IP címét. A dinamikus tartalom szerverek esetén szeretnénk még azt is eltárolni, hogy használnak-e cache-elést az adatbázis elérések minimalizálására. A klienseknek alapvetően tehát csak az alkalmazásszervert kell elérniük, hiszen az állítja össze a tartalmat. Az időszakos nagy terhelések tűrésére azonban a kliensek nem közvetlen kapcsolatban állnak az alkalmazásszerverrel, hanem egy terheléselosztón keresztül érik el azt. A terheléselosztónak van IP címe, tetszőlegesen sok alkalmazásszerverrel áll kapcsolatban és a redundancia érdekében legalább egy másik terheléselosztót is ismernie kell.

a) Készítsen el egy metamodellt, amely a fent leírt architektúra leírását lehetővé teszi! (4 pont)

b) Az architektúrát első körben a következőképpen képzeltük el. A kliens oldalon egy Androidos mobilos és egy webes kliens található. A webes kliens legalább Chrome 40, Firefox 33 és IE 10 verziót vár el. A szerver oldalon két terheléselosztó van, amelyek ismerik egymást, illetve mindegyik alkalmazásszervert. Jelenleg két alkalmazásszerver van. Az egyik egy cache-elt, a másik egy nem cache-elt dinamikus tartalom szerverrel áll kapcsolatban. Mindkét dinamikus tartalom szervernek külön adatbázisa van, az egyiknek Oracle, míg a másiknak MS SQL. Mindkét adatbázis külön diszket használ. Statikus tartalom szerverből kettő van, ezeket ismeri mindkét alkalmazásszerver. A statikus tartalom szervereknek is saját diszkjeik vannak. Készítse el az architektúra

példánymodelljét. Azon attribútumokat, amikre a leírás nem tért ki (IP címek, diszk méretek) valamilyen kitalált (de valósághű) értékkel töltse fel! (2 pont)

4.20 Otthoni hálózati eszközök (2015.06.03.)

A háztartásokban az elmúlt években igencsak megnőtt az internetet használó eszközök száma. A nappaliban lévő, modemre kötött egy gép mellé nagyobb családokban 1-2 laptop, okostévé ill. okostelefonok is társulnak. Szeretnénk egy olyan szoftveres megoldást szállítani, amivel modellezhető a mai modern otthonok hálózata.

Egy otthoni hálózat legalább egy, de akár több modemen keresztül csatlakozhat a külvilághoz. A modemnek van típusa (ami ADSL, kábel, mikrohullámú és mobilinternet lehet), gyártója és sorozatszáma. Egy modem egyszerre egy kapcsolat fenntartására képes. Egy modemkapcsolatot mindig jellemez a szolgáltató neve és a maximális le- és feltöltési sebesség. ADSL kapcsolat esetén még nyilván kell tartani a központtól becsült távolságot, mikrohullámú kapcsolat esetén a jelerősséget (0-100-ig terjedő szám) és mobilinternet esetén a kapcsolat típusát (ami GPRS, EDGE, 3G, HDSPA, HDSPA+ vagy LTE lehet). A modemhez pontosan egy hálózati eszköz köthető: ez lehet vezetékes router, access point, switch és hálózati kártya, utóbbi lehet vezetékes és vezeték nélküli is. Mindegyik hálózati eszköznek van gyártója, típusa, sorozatszáma és MAC címe. A fenti eszközöknek pontosan egy bemenetük van, azonban a routerhez, a switch-hez és az access point-hoz több eszköz is csatlakozhat. A kábeles eszközöket jellemzi egy portszám, a vezeték nélkülieket pedig a maximálisan felépíthető kapcsolatok száma. A hálózati kártyát minden esetben egy szórakoztatóelektronikai eszköz tartalmazza, ami lehet például PC, laptop, okostelefon vagy okostévé.

a) Készítsen el egy UML metamodellt, ami a fent leírt architektúra leírását lehetővé teszi! (3 pont)

b) Modellezze a következő otthoni hálózatot: a lakók 20 MBit/s sebességű ADSL kapcsolatra fizettek elő, amely modemjét egy PC-hez kötötték. A PC másik hálózati kártyájára egy D-Link access pointot csatlakoztattak, amihez maximálisan 20 eszköz csatlakozhat. Jelenleg csak két eszköz csatlakozik hozzájuk: egy okostévé (ami képes kábeles és WiFi-s kapcsolatra is), illetve egy okostelefon, ami párhuzamosan egy LTE kapcsolatot is fenntart. (2 pont)

c) A szoftver időközben elkészült, azonban több felhasználó is panaszkodott arra, hogy a szoftverrel nem lehet modellezni a WiFi routereket, illetve azt az esetet, amikor okostelefonról osztják meg az internetkapcsolatot. Hogyan javítaná ezeket a hiányosságokat? (1 pont)

4.21 Közösségi kerékpármegosztó (2015.06.10.)

Évről-évre egyre több nagyváros nyitja meg a kapuit a közösségi kerékpármegosztó rendszerek előtt. Az ilyen rendszerek lényege, hogy fix pontokon (dokkoló állomásokon) kerékpárok vannak elhelyezve, ezeket onnan lehet elvinni és hasonlóan ilyen pontokra kell használat után visszajuttatni. A rendszernek két alapeleme van: az egyik a kerékpár, a másik pedig a dokkoló állomás, ahol a kerékpárok találhatók. A kezdeti bevezetések után hamar felismerték, hogy a felhasználási mód felhasználónként igen eltérő: van, aki csak rövidebb ideig szeretne kényelmes tempóban közlekedni, van, aki gyorsan szeretne haladni és van, aki inkább azt részesíti előnyben, ha a kerékpár rendelkezik egy beépített motorral, ami az emelkedőkön használható. Így tehát kölcsönözhető városi kerékpár, országúti kerékpár illetve elektromos kerékpár. Minden kerékpáron van egy fedélzeti számítógép, ami tárolja, hogy milyen típusú a kerékpár, ki a gyártója, mi az azonosítója, kerékátmérője illetve tömege és teherbírása. A városi kerékpárok kerékátmérője kivétel nélkül 24”, az országúti kerékpárok tömege pedig mindig kisebb, mint 10 kg. Emellett a fedélzeti számítógép folyamatosan nyilvántartja, hogy mely állomáson tartózkodik a kerékpár illetve mi a kerékpár aktuális GPS koordinátája (szélesség és hosszúság), illetve tárolja az utolsó GPS koordinátához tartozó időbélyeget. A dokkoló állomásnak is több típusa van, azonban ezt úgy tervezték, hogy a későbbiekben is lehessen modulárisan változtatni. A dokkoló állomáshoz mindig tartozik egy kártyaleolvasó és egy segélyhívó modul. Emellett tartozhat hozzá fizető modul, a rendszerrel kapcsolatos tájékoztató alrendszer illetve turisztikai tájékoztató alrendszer. A

dokkoló állomáson egyszerre bármennyi kerékpár tárolható, azonban csak egy előre meghatározott számú kerékpár lehet töltőn (a rendszer mindig tudja, hogy melyik kerékpár hol van, illetve hogy melyik van töltés alatt). A felhasználóhoz egy RFID chippel ellátott kártya tartozik, amely tudja a saját azonosítóját. Az azonosítóhoz egy adatbázisban tároljuk a felhasználó nevét, e-mail címét, lakcímét és a felhasználó telefonszámát. Emellett minden felhasználóhoz tároljuk, hogy milyen típusú előfizetése van (lehetőségek: bronz, ezüst, arany). Egy felhasználó csak egy kerékpárt vihet el egyszerre.

a) Készítsen el egy UML metamodellt, ami a fent leírt architektúra leírását lehetővé teszi! Ügyeljen arra, hogy csak az informatikai rendszer elemeit modellezze! (3 pont)

b) Modellezze a következő rendszert: adott két dokkoló állomás, az egyiken két országúti a másikon két elektromos kerékpár van éppen. Mindkét állomáson csak egy kerékpár van töltés alatt. Két felhasználó van a rendszerben, az egyik éppen nem kölcsönöz kerékpárt, a másik viszont egy elektromos kerékpárt vezet. (2 pont)

c) A rendszer időközben elkészült, azonban több felhasználó is azt a visszajelzést adna, hogy szeretnének egyszerre több kerékpárt is elvinni. Hogyan módosítaná a modellt ahhoz, hogy az ezüst előfizetéssel rendelkező felhasználók 2, míg az arany előfizetéssel rendelkező felhasználók 5 kerékpárt is elvihessenek egy időben? (1 pont)