**Alternatív, konkurens modellkapcsolatok kialakítása**

Bócz Viktor, KJE

# Bevezető

Az általam kidolgozott vhcapp nevű rendszermodell egy olyan kapcsolatháló, amely egy kórházi időpontfoglalást hívatott szimulálni. Ez a rendszer szükségszerűen többrétegű, hiszen az időpontfoglalás folyamata több részfolyamatból és stációból tevődik össze. Ezt a többrétegűséget értelmezhetjük vertikális többrétegűségnek is.

A célja ennek a beadandónak, hogy ez a projektmunka kibővüljön a vertikális többrétegűségi dimenzión kívül egy alternatív, konkurens modellkapcsolati dimenzióval is. Ennek elérése érdekében a beadandó alanya a „Domain model” réteg lesz. Ezt a réteget önkényesen választottam ki, vezérlő elvnek azt a meggyőződést választottam, hogy ezen réteggel lesz a legegyszerűbb modellkapcsolati alternatívát nyújtani. Továbbra is az [UML](https://ceur-ws.org/Vol-1078/paper1.pdf) modellező nyelvet fogom használni ennek elérése érdekében.

Érdemes ismertetnünk az értelmezési tartományt is, így először a relációkat szimbolizáló nyilak jelentésére fogok kitérni:

* Öröklődés (Is-a relationship):

Amennyiben egy osztály rendelkezik ősosztállyal, abban az esetben a Generalizationt kell használjuk. (Példa: Person 🡨 Patient – vö. 1. ábra)

A képen diagram látható

Automatikusan generált leírás

1. ábra – Öröklődés, vhcapp: Domain Models page

* [Aggregation](https://www.uml-diagrams.org/aggregation.html) ([Has-a relationship](https://www.bestprog.net/en/2020/02/27/c-types-of-relationships-between-classes-is-a-has-a-uses-examples-aggregation-composition/)):

Amennyiben egy osztály rendelkezik egy olyan property-vel, amely egy másik osztályunk példányát tartalmazza és ezen példány megléte **nem elengedhetetlen** az osztályunk működésében. Itt az Aggregation arrow-t kell használnunk. (Példa: Patient 🡨 Appointment – vö. 2. ábra)

A képen diagram, sematikus rajz látható

Automatikusan generált leírás

2. ábra – Aggregation, vhcapp: Domain Models page

Az aggregation reláció nem csupán azt jelenti ebben az értelmezési tartományban, hogy az alanyául szolgáló entitás nem létezhet anélkül az attribútum nélkül, amelyre a reláció vonatkozik. **Az aggregáció annak a lehetőségét is magába foglalja, hogy az alanynak kontrollja van az aggregáció tárgyául szolgáló attribútum felett.** Példának okáért a Patient entitásnak kontrollja van a lefoglalt időpontok felett. Ugyanez érvényesül a többi entitás aggregation relációival kapcsolatban is.

* [Composition](https://www.uml-diagrams.org/composition.html) (Has-a relationship):

Amennyiben egy osztály rendelkezik egy olyan property-vel, amely egy másik osztályunk példányát tartalmazza és ezen példány megléte **elengedhetetlen** az osztályunk működésében. Itt a Composition arrow-t kell használnunk. (Példa: Appointment 🡨 Doctor – vö. 3. ábra)

A képen diagram látható

Automatikusan generált leírás

3. ábra – Composition, vhcapp: Domain Models page

# Eredeti modellkapcsolat

Elsődlegesen fontosnak tartom bemutatni a meglévő modellkapcsolatunkat, melyet az alábbi képpel illusztrálom:

A képen diagram látható

Automatikusan generált leírás

4. ábra – Domain Models kapcsolatok, vhcapp: Domain Models page

A fentiek tükrében megállapíthatóak az alábbi állítások a modellkapcsolatról – vö. 4. ábra:

* Mind a Patient, mind a Doctor örökli a Person tulajdonságait, hiszen a Person osztályból származtatjuk mindkét osztályt. Az Appointment nem tagja semmilyen öröklődési viszonyrendszernek.
* Az Appointment entitás **nem létezhet** \_doctor entitás nélkül, elvégre a való életben sincsen olyan orvosi időpont, ahol nincs kezelőorvos. Ennek okán használtam itt a composition relációt. Ugyanez nem mondható el a pácienssel kapcsolatos relációról, elvégre egy szabad orvosi időpontnak sincs előre definiált páciense.
* A páciensnek lehetnek előjegyzett időpontjai, így a Patient entitásnál megtalálható appointments lista/array property rendelkezhet 0, 1, vagy akár több elemmel is. Az aggregation relációt alkalmaztam itt, hiszen a páciens entitásunk létrejöhet érvényes időpont nélkül, **nem elengedhetetlen**, hogy legyen érvényes foglalásunk.

A fenti megállapítások erősen véleményesek és ez nem is meglepő, hiszen ez a domain réteg a teljesség igénye nélkül készült. A soron következő fejezet ennek a modellkapcsolatnak az alternatíváit fogja bemutatni, mely konkurenciája vagy kiegészítője is lehet az eddig létrehozottaknak.

# Konkurens modellkapcsolatok

## Részben partneri viszonyra épülő és azt feltételező modellkapcsolat

Ez a kapcsolati hálózat némileg kiegészíti az eredeti megvalósítást, alternatív megközelítést részben tartalmaz csak.

Ami a kapcsolati hálónk szereplőit illeti, két új entitás is bekerült a viszonyrendszerbe: a DateTime és a Health Center. A [DateTime](https://docs.python.org/3/library/datetime.html) egy, a legtöbb programozási nyelvben megtalálható, beépített adattípus. Ezzel az adattípussal kívántam megjeleníteni azokat a lehetséges időpontokat, melyeket a ­Health Center entitásunk felkínál a páciensek számára. A Health Center pedig maga a helyszín és jogi entitás, mely lehetővé teszi az orvosi időpont megvalósulását.

A képen diagram látható

Automatikusan generált leírás

5. ábra – Partly partner-based model structure, alternative\_vhcapp: Partly partner-based model structure page

Az ábránk alapján ismertetném az alábbi megállapításokat tehetjük – vö. 5. ábra:

* Mind a Patient, mind a Doctor örökli a Person tulajdonságait, hiszen a Person osztályból származtatjuk mindkét osztályt. Az Appointment, a HealthCenter és a DateTime nem tagja semmilyen öröklődési viszonyrendszernek.
* Ami a composition relációkat illeti:
  + Az Appointment entitás **nem létezhet** \_doctor, \_patient és \_date entitások nélkül, elvégre a való életben sincsen olyan orvosi időpont, ahol nincs kezelőorvos, páciens és a dátum, mely alkalommal megvalósul az ellátás. A Health Center entitás megléte az Appointment kapcsán nem elengedhetetlen, hiszen a vizsgálat megvalósulhat külső helyszínen is.
* Ami pedig az aggregation relációkat illeti:
  + A Health Center entitásunk **létezhet** Doctor, Patient, Appointment és DateTime (\_possibleDates) entitások nélkül is, hiszen akár egy felújítási munkálat/szünetelt gazdasági tevékenység során az entitás léte nem válik fenntarthatatlanná/értelmetlenné.
  + A páciensnek lehetnek előjegyzett időpontjai, így a Patient entitásnál megtalálható appointments lista/array property rendelkezhet 0, 1, vagy több elemmel is. Az aggregation relációt alkalmaztam itt, hiszen a páciens entitásunk létrejöhet érvényes időpont nélkül, **nem elengedhetetlen** a megléte.
  + A Doctor entitás **létezhet** aktív orvosi időpontok nélkül, például egy olyan esetben, ha átmenetileg szünetelteti az együttműködését az egészségügyi központtal (Health Center), de annak alkalmazásában marad.

Attól válik ez a modell részben partneri alapokon nyugvó rendszermodellé, hogy a Health Centernek kontrollja van a lefoglalt és a szabad időpontok felett is, ennek okán az nem csupán az orvos felelősségi körébe tartozik. Úgy képzelem, hogy egy **alvállalkozói rendszerben** ehhez hasonló működést figyelhetnénk meg.

## Partneri viszonyra épülő és azt feltételező modellkapcsolat

A most bemutatásra kerülő kapcsolati rendszer valós alternatívát tud nyújtani az eredeti és a részben partneri viszonyrendszerre épülő modellekhez képest.

A képen diagram látható

Automatikusan generált leírás

6. ábra - Partner-based model structure, alternative\_vhcapp: Partner-based model structure page

Az ábránk alapján ismertetném az alábbi megállapításokat tehetjük – vö. 6. ábra:

* Mind a Patient, mind a Doctor örökli a Person tulajdonságait, hiszen a Person osztályból származtatjuk mindkét osztályt. Az Appointment, a HealthCenter és a DateTime nem tagja semmilyen öröklődési viszonyrendszernek.
* Ami a composition relációkat illeti:
  + Az Appointment entitás **nem létezhet** \_doctor, \_patient és \_date entitások nélkül, elvégre a való életben sincsen olyan orvosi időpont, ahol nincs kezelőorvos, páciens és a dátum, mely alkalommal megvalósul az ellátás. A Health Center entitás megléte az Appointment kapcsán nem elengedhetetlen, hiszen a vizsgálat megvalósulhat külső helyszínen is.
* Ami pedig az aggregation relációkat illeti:
  + A Health Center entitásunk **létezhet** Doctor entitás nélkül, hiszen akár egy felújítási munkálat/szünetelt gazdasági tevékenység során az entitás léte nem válik fenntarthatatlanná/értelmetlenné.
  + A páciensnek lehetnek előjegyzett időpontjai, így a Patient entitásnál megtalálható appointments lista/array property rendelkezhet 0, 1, vagy több elemmel is. Az aggregation relációt alkalmaztam itt, hiszen a páciens entitásunk létrejöhet érvényes időpont nélkül, **nem elengedhetetlen** a megléte.
  + A Doctor entitás **létezhet** aktív orvosi időpontok és lehetséges időpontok nélkül is, például egy olyan esetben, ha átmenetileg szünetelteti az együttműködését az egészségügyi központtal (Health Center), de annak alkalmazásában marad.

Attól válik ez a modell partneri alapokon nyugvó rendszermodellé, hogy a Health Centernek nincs kontrollja a lefoglalt és a szabad időpontok felett, ennek okán az csupán az orvos felelősségi körébe tartozik. Itt ugyanis az egészségügyi központ csupán azt szabhatja meg, hogy mely orvosokkal kíván együttműködni, illetve a vizsgálat helyét biztosítja. Úgy képzelem, hogy egy **vállalkozói rendszerben** ehhez hasonló működést figyelhetnénk meg.

## Vezényléses viszonyra épülő és azt feltételező modellkapcsolat

A most bemutatásra kerülő kapcsolati rendszer valós alternatívát tud nyújtani az eddig felvázolt viszonyrendszerekhez képest.

A képen diagram látható

Automatikusan generált leírás

7. ábra – Demand-based model structure, alternative\_vhcapp: Demand-based model structure page

Az ábránk alapján ismertetném az alábbi megállapításokat tehetjük – vö. 7. ábra:

* Mind a Patient, mind a Doctor örökli a Person tulajdonságait, hiszen a Person osztályból származtatjuk mindkét osztályt. Az Appointment, a HealthCenter és a DateTime nem tagja semmilyen öröklődési viszonyrendszernek.
* Ami a composition relációkat illeti:
  + Az Appointment entitás **nem létezhet** \_doctor, \_patient és \_date entitások nélkül, elvégre a való életben sincsen olyan orvosi időpont, ahol nincs kezelőorvos, páciens és a dátum, mely alkalommal megvalósul az ellátás. A Health Center entitás megléte az Appointment kapcsán nem elengedhetetlen, hiszen a vizsgálat megvalósulhat külső helyszínen is.
* Ami pedig az aggregation relációkat illeti:
  + A Health Center entitásunk **létezhet** Doctor, Patient, Appointment és DateTime (\_possibleDates) entitások nélkül is, hiszen akár egy felújítási munkálat/szünetelt gazdasági tevékenység során az entitás léte nem válik fenntarthatatlanná/értelmetlenné.
  + A páciensnek lehetnek előjegyzett időpontjai, így a Patient entitásnál megtalálható appointments lista/array property rendelkezhet 0, 1, vagy több elemmel is. Az aggregation relációt alkalmaztam itt, hiszen a páciens entitásunk létrejöhet érvényes időpont nélkül, **nem elengedhetetlen** a megléte.

Attól válik ez a modell vezényléses alapokon nyugvó rendszermodellé, hogy a Health Centernek kontrollja van a lefoglalt és a szabad időpontok felett is, ennek okán az nem tartozik az orvos felelősségi körébe. Történetesen ebben a rendszerben semmi nem tartozik az orvos felelősségi körébe, egyedül a szakértelmét értékesíti a vizsgálat során. Úgy képzelem, hogy egy **munkáltató-beosztott** rendszerben ehhez hasonló működést figyelhetnénk meg.

# Összefoglaló

Az általam felvázolt modellek bizonyítják, hogy egy adott feladathalmazt eltérő megközelítésekkel oldhatunk meg, így van tere a megrendelőnek a saját elképzeléseit érvényesíteni egy fejlesztési folyamatban. Ez kölcsönösen előnyös, hiszen a megrendelő azt kapja, amiért fizet, a végrehajtó pedig megkapja, amiért dolgozik, az pedig az elvégzett munkáért járó fizetség. A fenti példából azt is leszűrhetjük, hogy az UML alkalmas „vivőanyag” ezen célok megvalósulásához.

# Tartalomjegyzék

[Bevezető 1](#_Toc143515502)

[Eredeti modellkapcsolat 3](#_Toc143515503)

[Konkurens modellkapcsolatok 4](#_Toc143515504)

[I. Részben partneri viszonyra épülő és azt feltételező modellkapcsolat 4](#_Toc143515505)

[II. Partneri viszonyra épülő és azt feltételező modellkapcsolat 6](#_Toc143515506)

[III. Vezényléses viszonyra épülő és azt feltételező modellkapcsolat 7](#_Toc143515507)

[Összefoglaló 9](#_Toc143515508)

[Tartalomjegyzék 9](#_Toc143515509)

[Ábrajegyzék 10](#_Toc143515510)

[Irodalomjegyzék 11](#_Toc143515511)

[Rövidítésjegyzék 11](#_Toc143515512)

# Ábrajegyzék

[1. ábra – Öröklődés, vhcapp: Domain Models page 2](#_Toc132648689)

[2. ábra – Aggregation, vhcapp: Domain Models page 2](#_Toc132648690)

[3. ábra – Composition, vhcapp: Domain Models page 3](#_Toc132648691)

[4. ábra – Domain Models kapcsolatok, vhcapp: Domain Models page 3](#_Toc132648692)

[5. ábra – Partly partner-based model structure, alternative\_vhcapp: Partly partner-based model structure page 5](#_Toc132648693)

[6. ábra - Partner-based model structure, alternative\_vhcapp: Partner-based model structure page 6](#_Toc132648694)

[7. ábra – Demand-based model structure, alternative\_vhcapp: Demand-based model structure page 8](#_Toc132648695)

# Irodalomjegyzék

BestProgisch. (2020. február 27). *BestProg*. Forrás: www.bestprog.net: https://www.bestprog.net/en/2020/02/27/c-types-of-relationships-between-classes-is-a-has-a-uses-examples-aggregation-composition/

community, P. (2023. április 17). *Python official documentation*. Forrás: https://docs.python.org/: https://docs.python.org/3/library/datetime.html

Fakhroutdinov, K. (dátum nélk.). *uml-diagrams*. Forrás: www.uml-diagrams.org: https://www.uml-diagrams.org/aggregation.html

Fakhroutdinov, K. (dátum nélk.). *uml-diagrams*. Forrás: https://www.uml-diagrams.org: https://www.uml-diagrams.org/composition.html

Gianna Reggio, M. L. (dátum nélk.). *What are the used UML diagrams?* Forrás: https://ceur-ws.org: https://ceur-ws.org/Vol-1078/paper1.pdf

# Rövidítésjegyzék

1. UML - Unified Modeling Language