Eötvös Loránd Tudományegyetem
Informatikai Kar

Komponens alapú szoftverfejlesztés

9. előadás

Grafikus felületű alkalmazások architektúrái

Giachetta Roberto

groberto@inf.elte.hu
http://people.inf.elte.hu/groberto

Grafikus felületű alkalmazások architektúrái

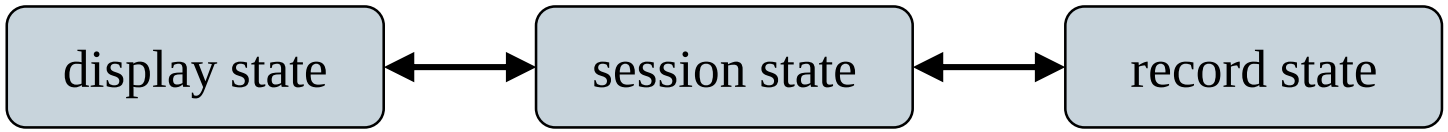
A grafikus felületű alkalmazás

- A *grafikus felületű* (*graphical user interface*) alkalmazások jelentős részét képezik a mai szoftvereknek
  - közvetlenül a felhasználóval állnak kapcsolatban, aki számára információkat jelenítenek meg, és feldolgozzák a felhasználói bemenetet, amihez *eseményvezérlés*t használnak
  - általánosan megfogalmazott grafikus elemekből (vezérlőkből) alkalmazás specifikus specifikus űrlapokat állítanak össze
- Grafikus felületű alkalmazások esetén a leggyakoribb felépítés a háromrétegű (3-tier) architektúra, amelyben elkülönül a nézet, a modell és a perzisztencia
  - a nézet tartalmazza az adatok megjelenítésének módját, valamint a felhasználói interakció feldolgozását

Grafikus felületű alkalmazások architektúrái

Adatok állapotai

- A háromrétegű alkalmazásokban az adatok három állapotban jelennek meg
  - *megjelenített állapot* (*display state*): a felhasználói felületen megjelenő tartalomként, amelyet a felhasználó módosíthat
  - *munkafolyamat állapot* (*session state*): a memóriában, amely mindaddig elérhető, amíg a program és felhasználója aktív
  - *rekord állapot* (*record state*): az adat hosszú távon megőrzött állapota az adattárban (perzisztenciában)



Grafikus felületű alkalmazások architektúrái

Adatok állapotai

- Az állapotok külön általában reprezentációt igényelnek, ezért az állapotok között *transzformáció*kat kell végeznünk
  - pl. objektumok közötti leképezés (*object-object mapping*), objektum-relációs leképezés (*object-relational mapping*)
- Az egyes állapotok kezelése történhet
  - *szinkron módon*: a két állapot mindig megegyezik
    - a megjelenített és munkafolyamat állapotnak célszerű megegyeznie, mivel a munkafolyamat állapoton hajtjuk végre a tevékenységeket
  - *aszinkron módon*: a két állapot általában különbözik, de adott ponton szinkronizálható

Grafikus felületű alkalmazások architektúrái

Háromrétegű architektúra specializációja

- A háromrétegű architektúrában a felhasználóval a nézet réteg tartja a kapcsolatot, amely ezért tartalmazza a *felületi logikát* (*UI logic*)
  - a felületi logika feladatai:
    - felhasználói interakció fogadása, feldolgozása, és továbbítása az üzleti logika számára
    - az adatok megjelenítése (a megjelentéshez szükséges konverzióval)
    - a megfelelő nézet előállítása, új nézetek létrehozása
  - a felületi logika és maga a megjelenítés két jól elhatárolható feladatkör, amelyet célszerű szeparálni
    - a megjelenítés elsősorban nem programozói, hanem tervezői feladatkör

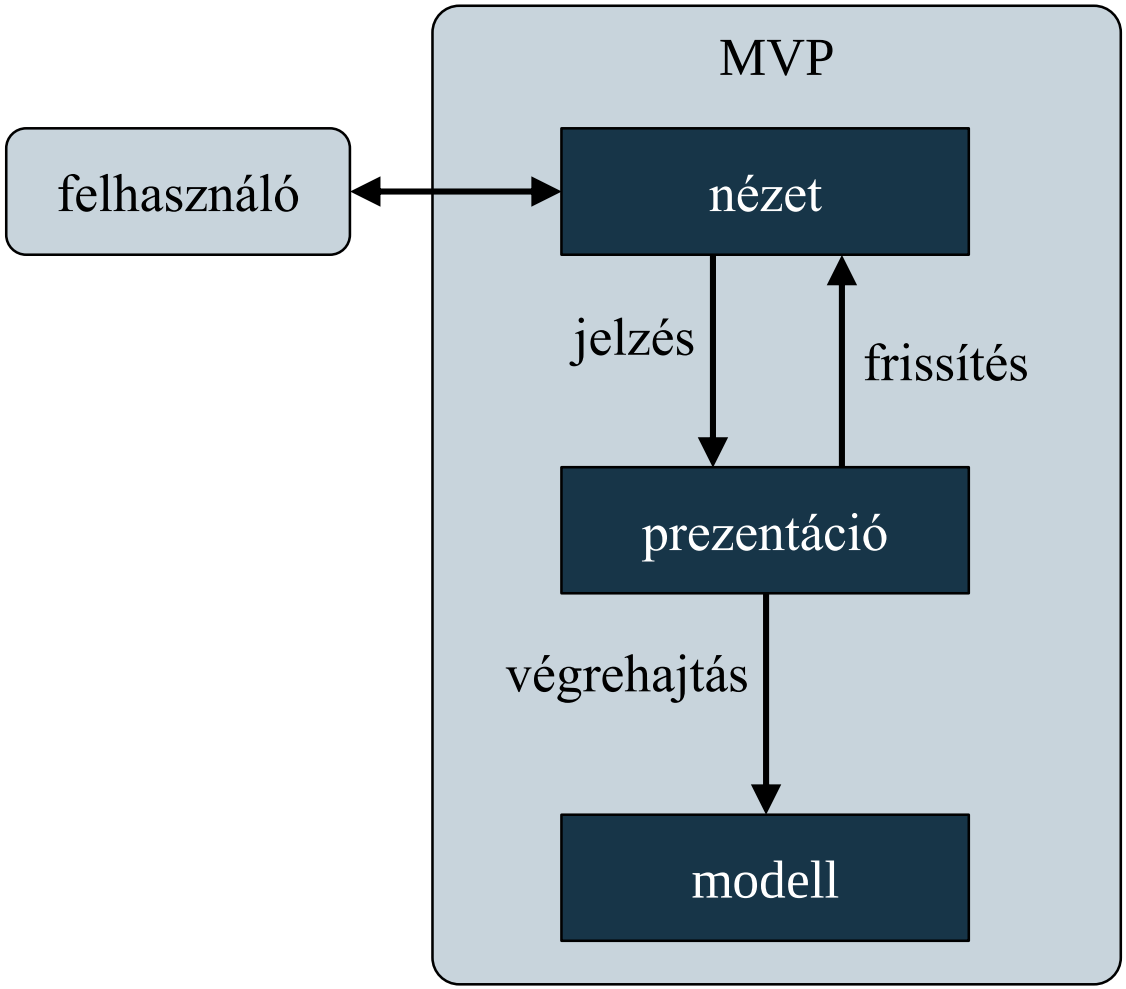
Grafikus felületű alkalmazások architektúrái

MVP architektúra

- A *Model-View-Presenter* (*MVP*) architektúra lehetőséget ad a felületi logika leválasztására egy *prezentáció* (*presenter*) számára
  - a *nézet* felel az adatok megjelenítéséért (nézetállapot előállításáért), valamint a felhasználó interakció fogadásáért, és továbbításáért a prezentáció számára
  - a *prezentáció* tartalmazza a felhasználói interakció feldolgozásáért felelős tevékenységeket, úgymint
    - továbbítja a kéréseket az üzleti logika számára
    - megadja az interakcióra válaszoló nézetet
  - a prezentáció ismeri a nézetet, a nézet azonban nem ismeri a prezentációt

Grafikus felületű alkalmazások architektúrái

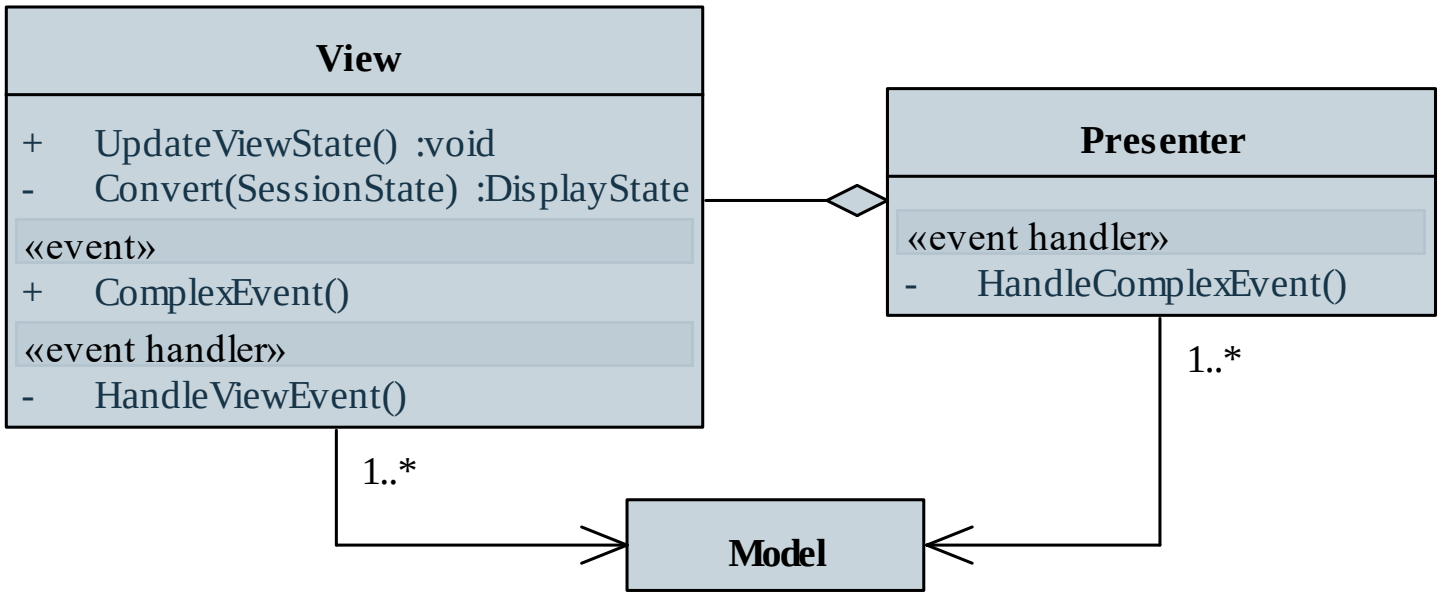
MVP architektúra



Grafikus felületű alkalmazások architektúrái

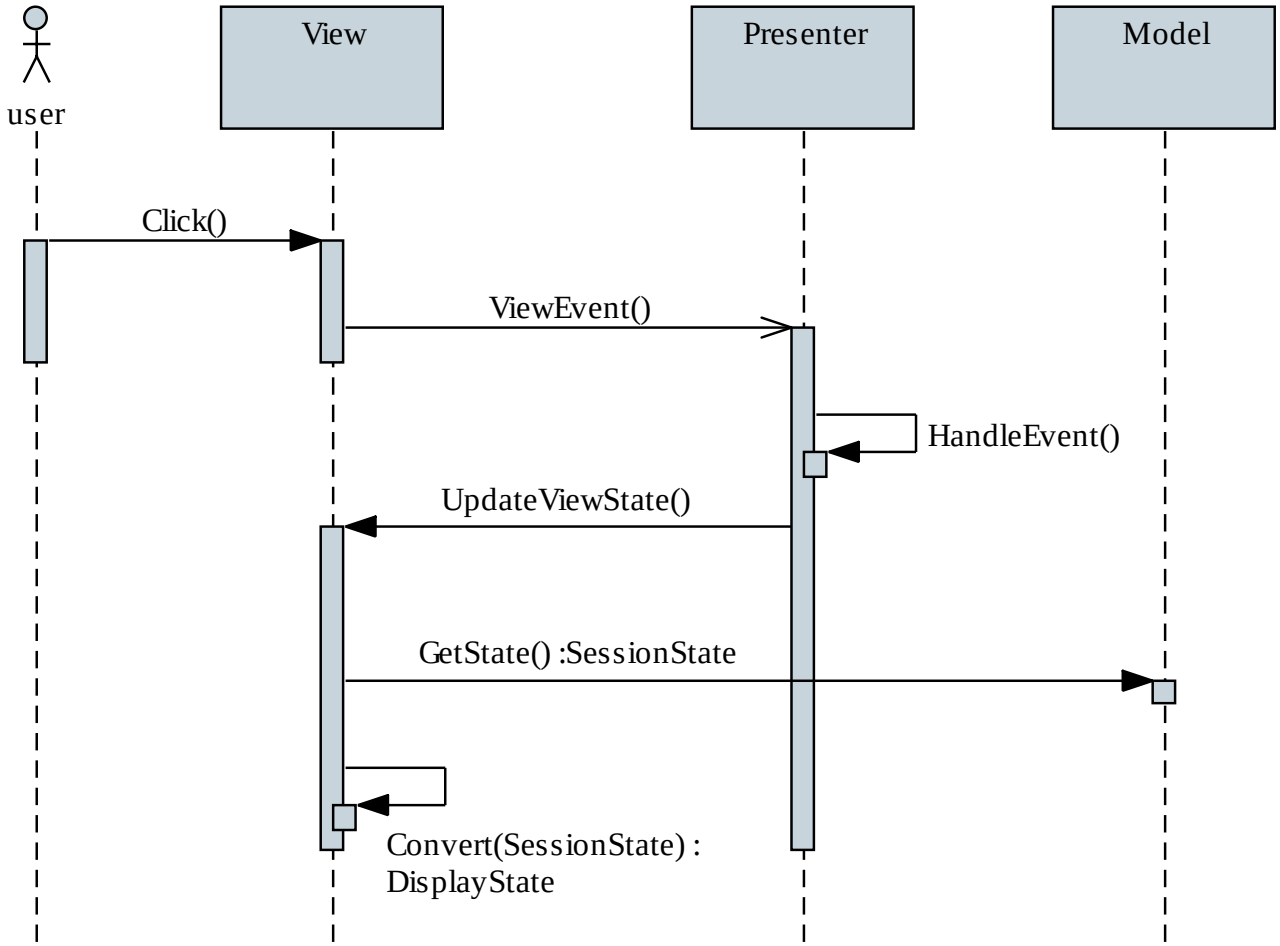
MVP architektúra (supervising controller)

- Az MVP architektúra két esete különböztethető meg:
  - *Supervising contoller* (*SC*): a nézet ismeri a modellt (le tudja kérni a munkafolyamat állapotot), és el tud végezni alapvető tevékenységeket



Grafikus felületű alkalmazások architektúrái

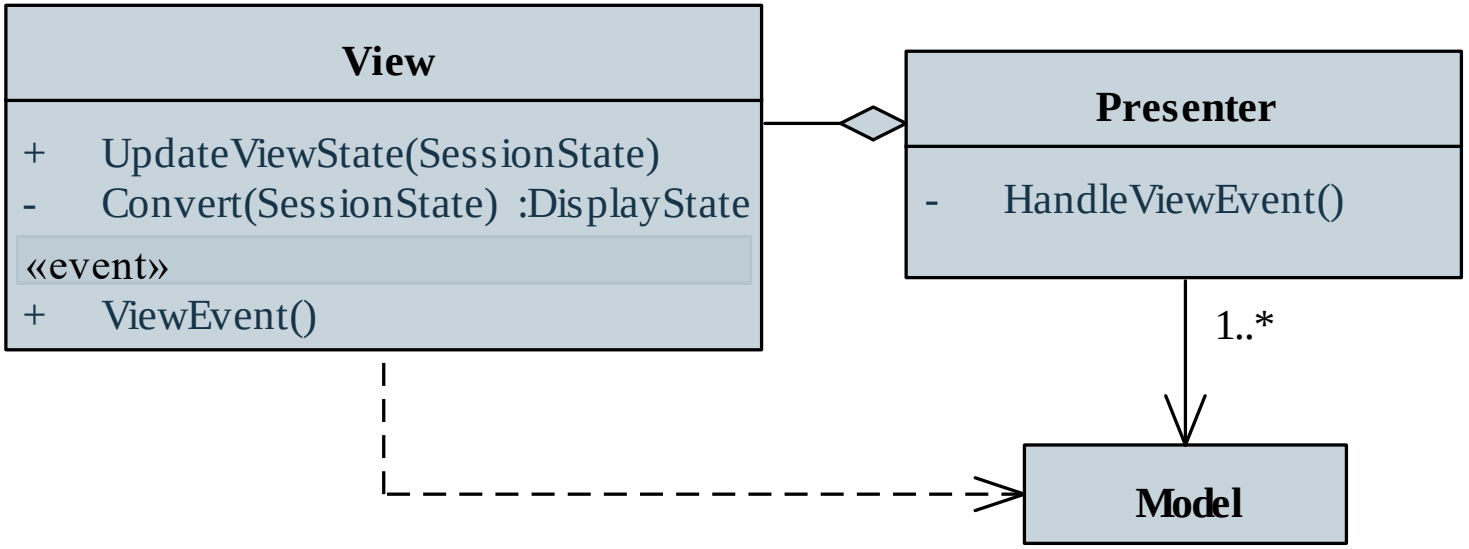
MVP architektúra (supervising controller)



Grafikus felületű alkalmazások architektúrái

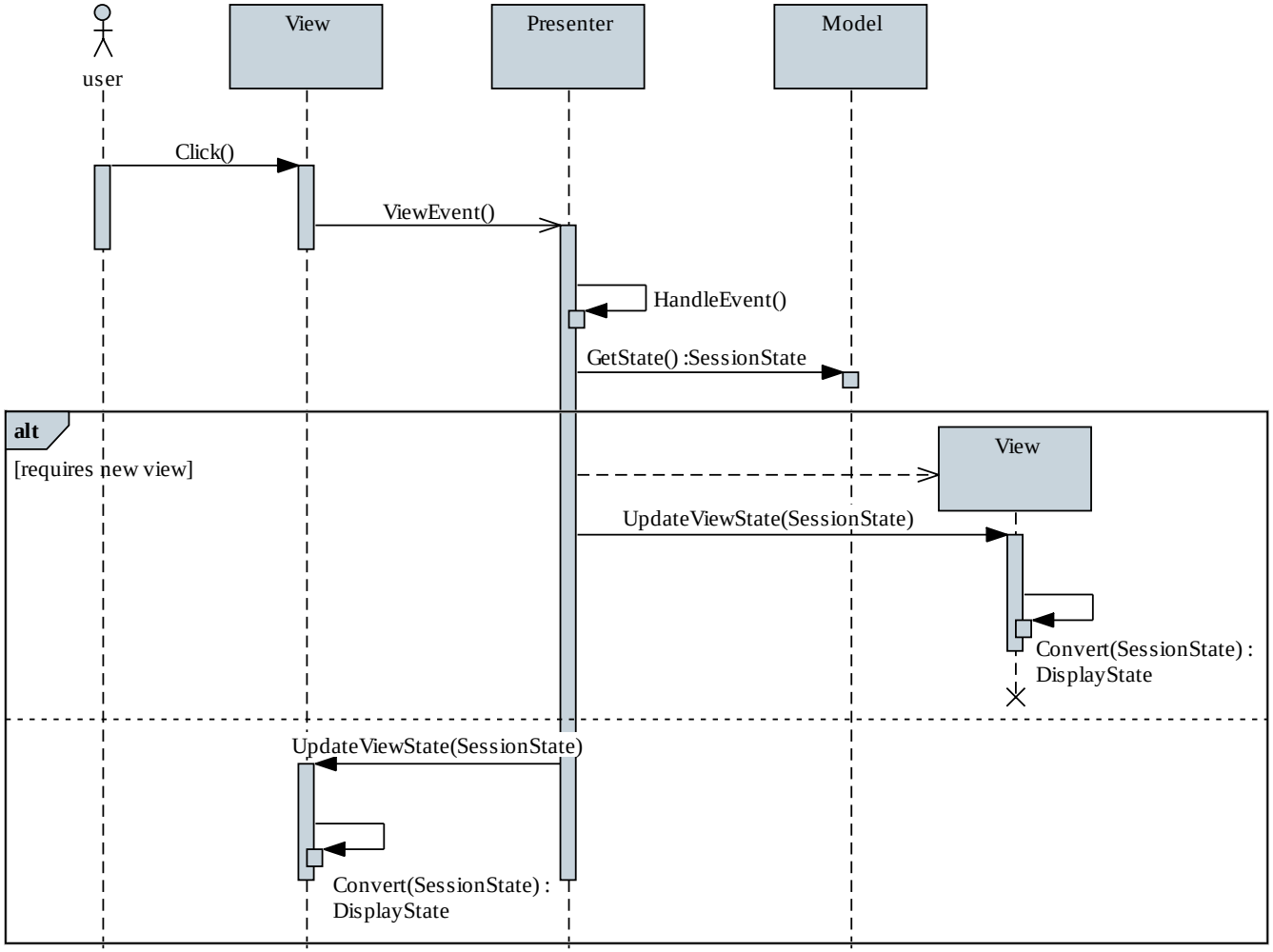
MVP architektúra (passive view)

- *Passive View* (*PV*): a nézet nem ismeri a modellt, így a prezentáció adja át a nézet számára a munkafolyamat állapotot
  - szinte minden tevékenységet a prezentáció végez
  - a nézet elsősorban grafikus vezérlők halmazának tekinthető



Grafikus felületű alkalmazások architektúrái

MVP architektúra (passive view)



Grafikus felületű alkalmazások architektúrái

MVP architektúra

- Előnyei:
  - a felületi logika leválasztható a nézetről, így a nézet ténylegesen nem lép interakcióba az üzleti logikával
  - a prezentáció a nézettől függetlenül tudja biztosítani új nézetek létrehozását és a munkafolyamat állapot manipulálását
  - a nézet független a prezentáció és a modell felületétől, így tetszőlegesen változtatható
  - a prezentáció és a nézet egymástól függetlenül tesztelhetőek
- Hátrányai:
  - a nézet továbbra is tartalmaz kódot (eseménykezelők formájában)
  - a megjelenített állapot frissítése manuálisan történik

Grafikus felületű alkalmazások architektúrái

Példa

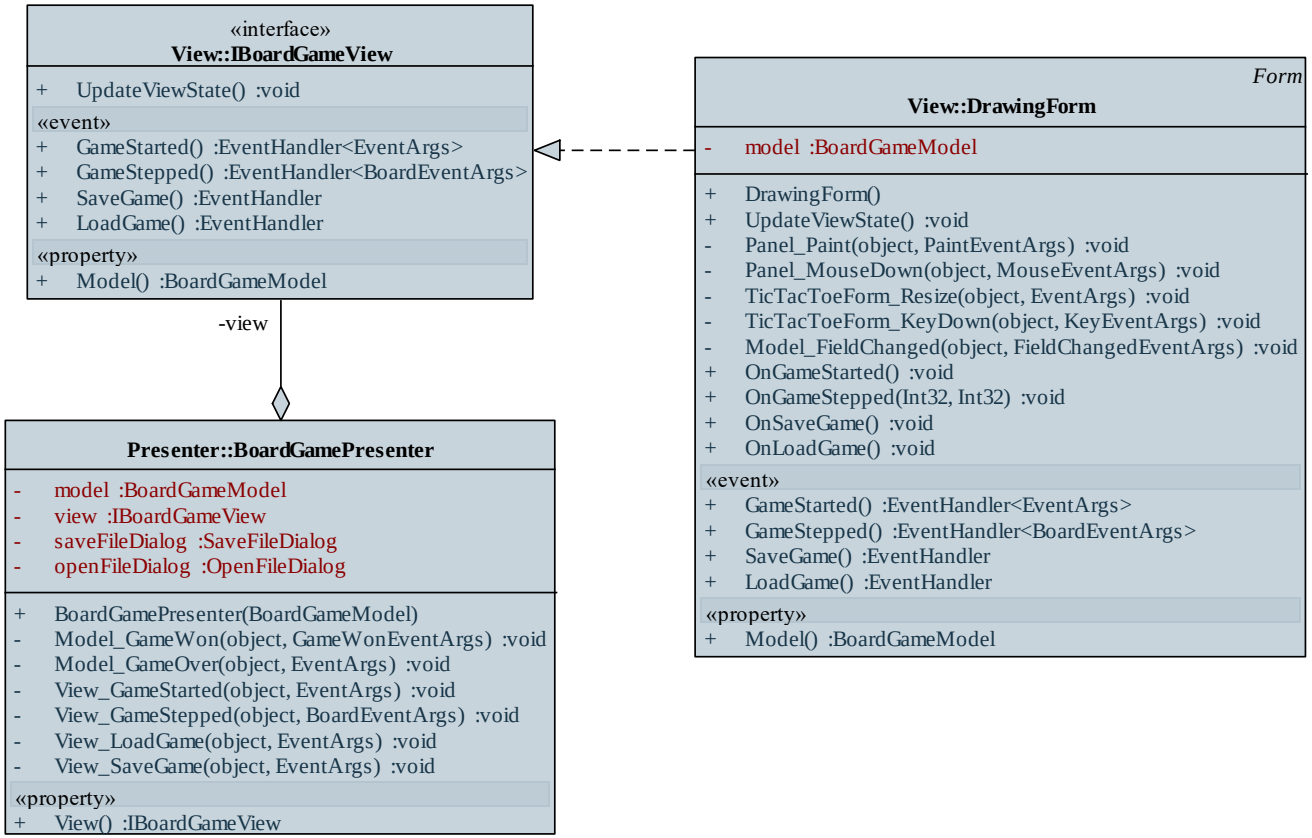
Feladat: Készítsünk egy táblajáték programot MVP SC architektúrában, amelyben két játékos küzdhet egymás ellen.

- a korábbi nézet tevékenységeit felbontjuk a nézet (**`IBoardGameView`**) és a prezentáció (**`BoardGamePresenter`**) között
  - a nézet interfésze egységes eseményeket definiál a játék tevékenységeire (**`GameStarted`**, **`GameStepped`**, …), valamint megadja a frissítés lehetőségét (**`UpdateViewState`**)
  - a nézet megvalósítása (**`DrawingForm`**) felel a tényleges események átalakításáért játékbeli eseményekké
  - a prezentáció kezeli a tevékenységeket, valamint a modell összetett eseményeit (**`GameWon`**, **`GameOver`**)

Grafikus felületű alkalmazások architektúrái

Példa

Tervezés:



Grafikus felületű alkalmazások architektúrái

Példa

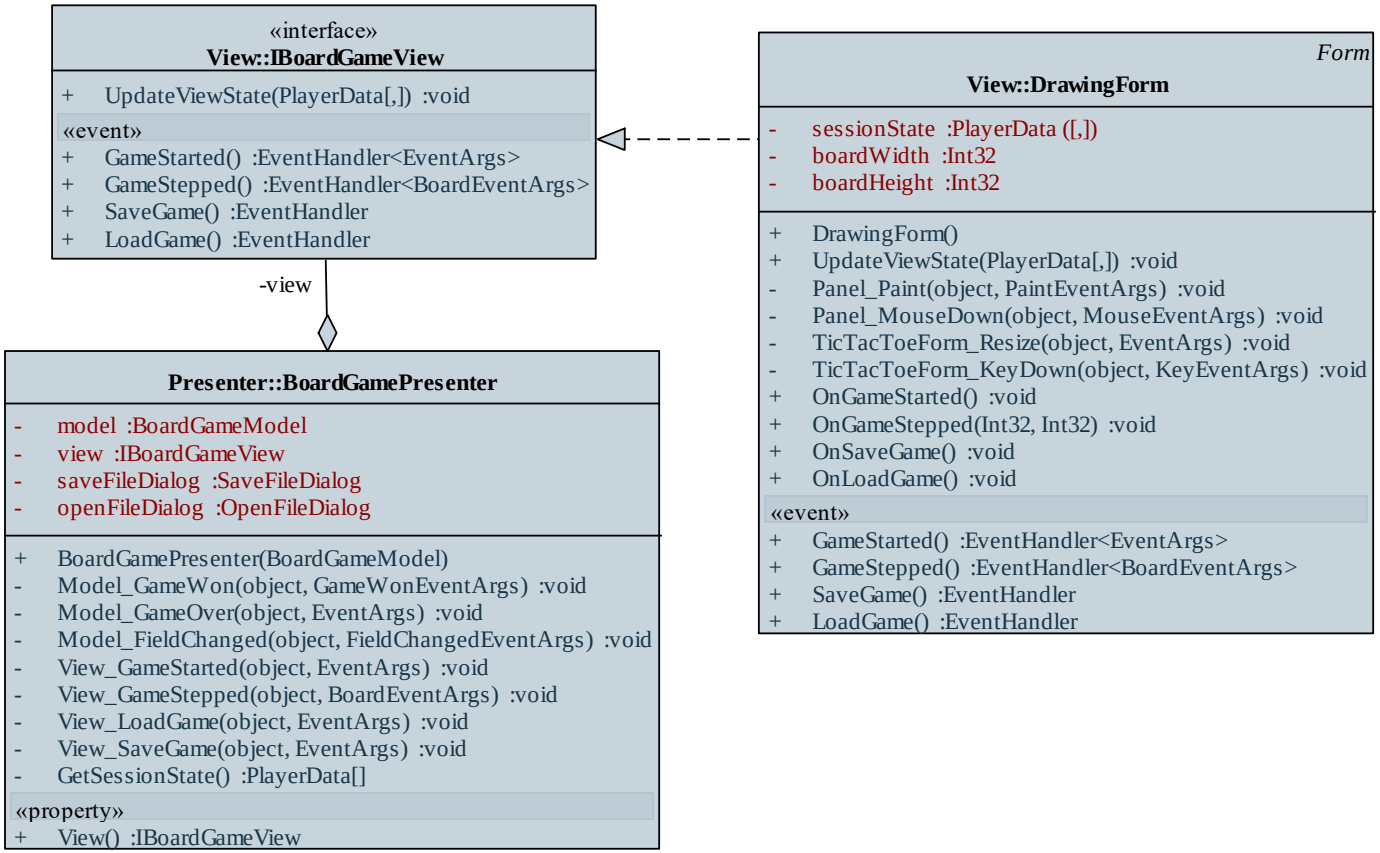
Feladat: Készítsünk egy táblajáték programot MVP PV architektúrában, amelyben két játékos küzdhet egymás ellen.

- a nézet (**`IBoardGameView`**) nem látja közvetlenül a modellt, így a prezentáció (**`BoardGamePresenter`**) kéri le a munkafolyamat állapotát (**`GetSessionState`**), és adja át a nézet számára
- a prezentáció veszi át a megmaradt modellbeli események kezelését a nézettől (**`Model_FieldChanged`**)

Grafikus felületű alkalmazások architektúrái

Példa

Tervezés:



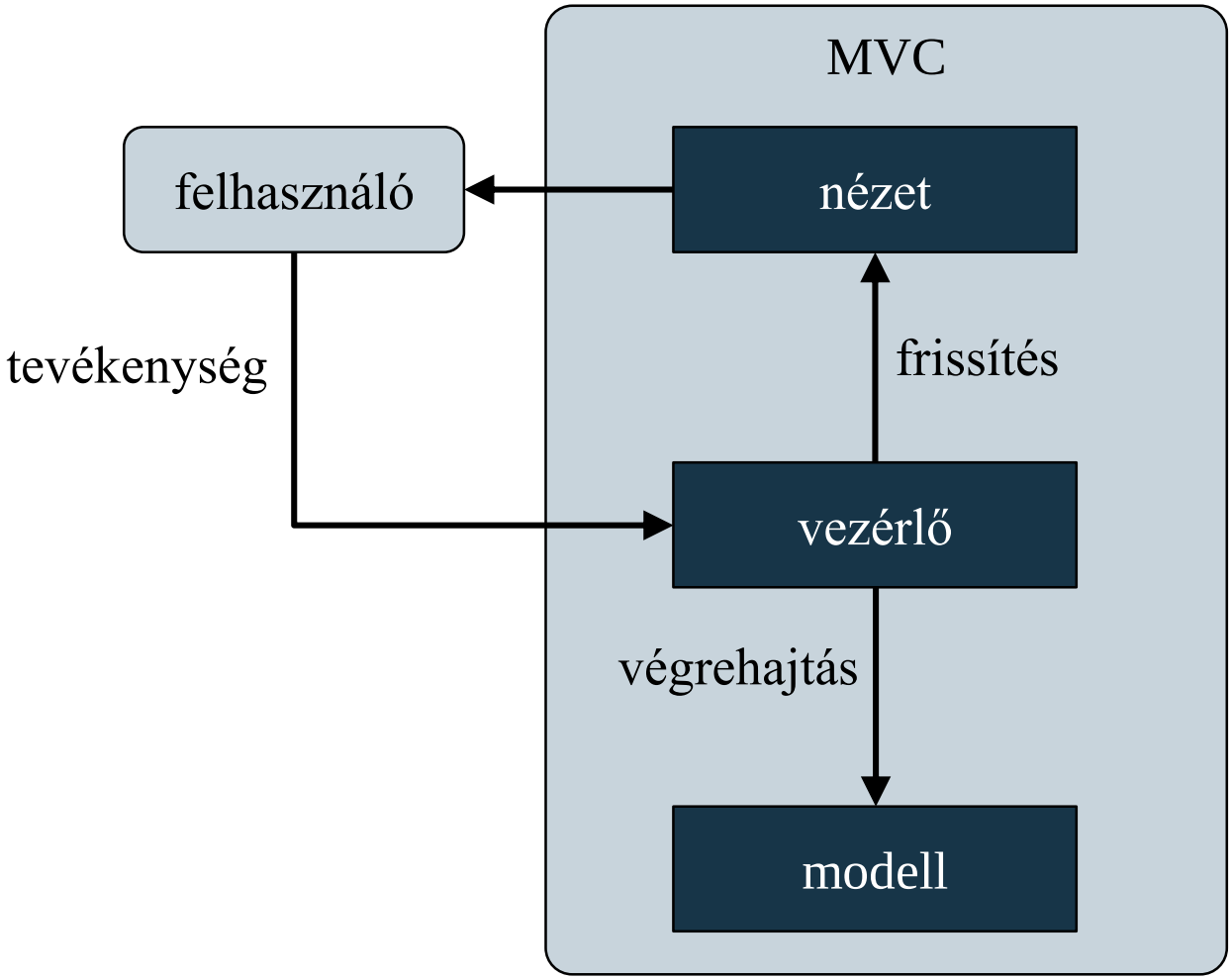
Grafikus felületű alkalmazások architektúrái

MVC architektúra

- A *Model-View-Controller* (*MVC*) architektúra egy külön *vezérlés*t (*contoller*) biztosít, amely kiszolgálja a felhasználói kéréseket
  - a *vezérlő* fogadja közvetlenül a kérést a felhasználótól, feldolgozza azt (a modell segítségével), majd előállítja a megfelelő (új) nézetet
    - így a nézet mentesül a kérések fogadásától, átalakításától
    - általában egy vezérlőhöz több nézet is tartozik
  - a *nézet* a felület (jórészt deklaratív) meghatározása, amely megjeleníti, szükség esetén átalakítja az adatokat
  - elsősorban webes környezetben népszerű, mivel könnyen leválasztható a nézet (HTML) a vezérléstől (URL)

Grafikus felületű alkalmazások architektúrái

MVC architektúra



Grafikus felületű alkalmazások architektúrái

MVC architektúra

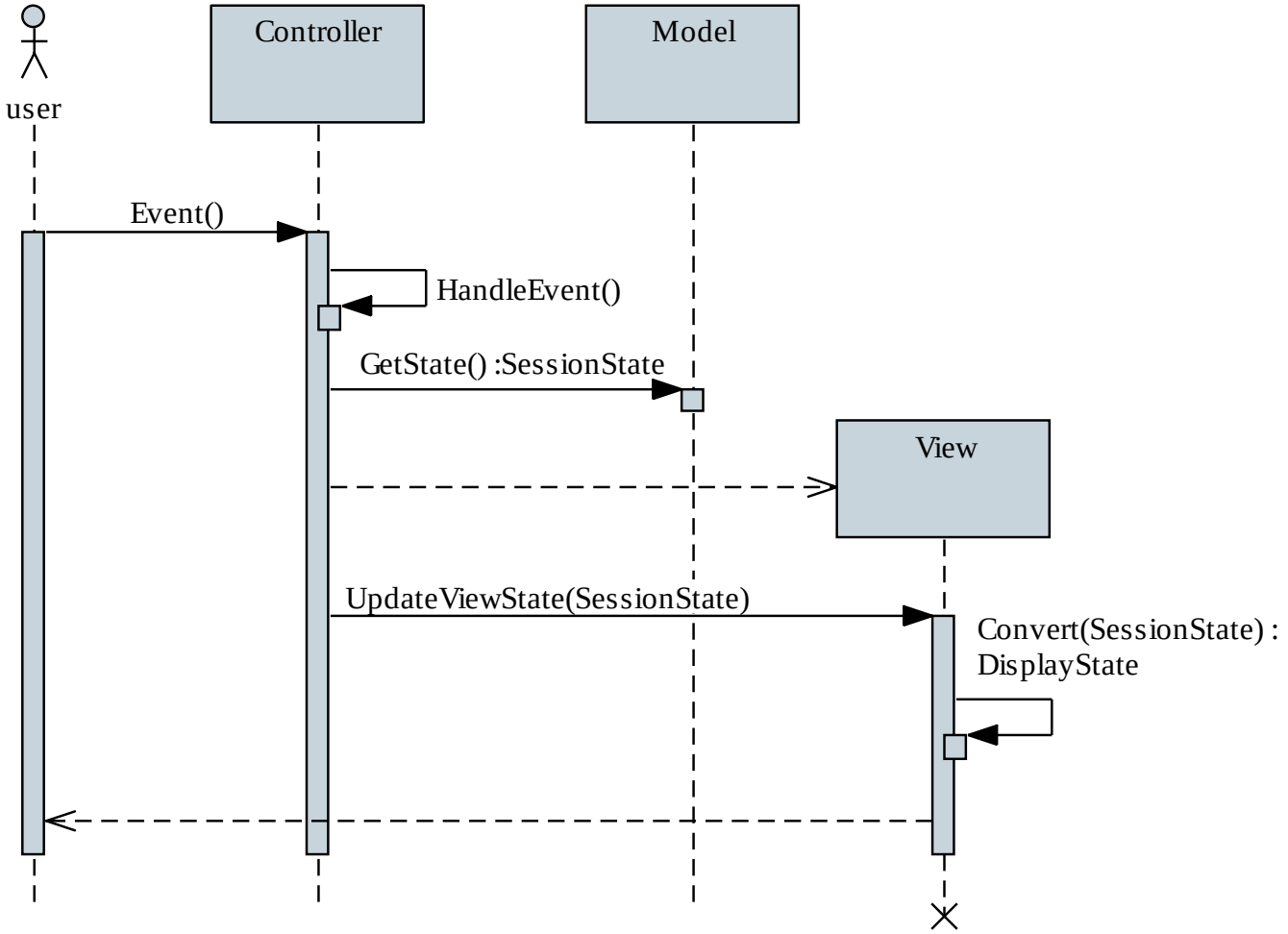
- Az MVC architektúra két esete különböztethető meg:
  - *lehívás alapú* (*pull-based*): ismeri a modellt, az adatokat a modelltől kéri le
  - *betöltés alapú* (*push-based*): a vezérlő adja át a nézetnek a megjelenítendő adatokat (ez a *nézet modell*je, vagy *view model*)

- Előnye:
  - a nézetnek nem kell foglalkoznia az események feldolgozásával, mivel azokat közvetlenül a vezérlő kapja meg

- Hátránya:
  - a nézet továbbra is felel az adatok megfelelő átalakításáért, tehát tartalmaz valamennyi logikát

Grafikus felületű alkalmazások architektúrái

MVC architektúra (betöltés alapú)



Grafikus felületű alkalmazások architektúrái

Példa

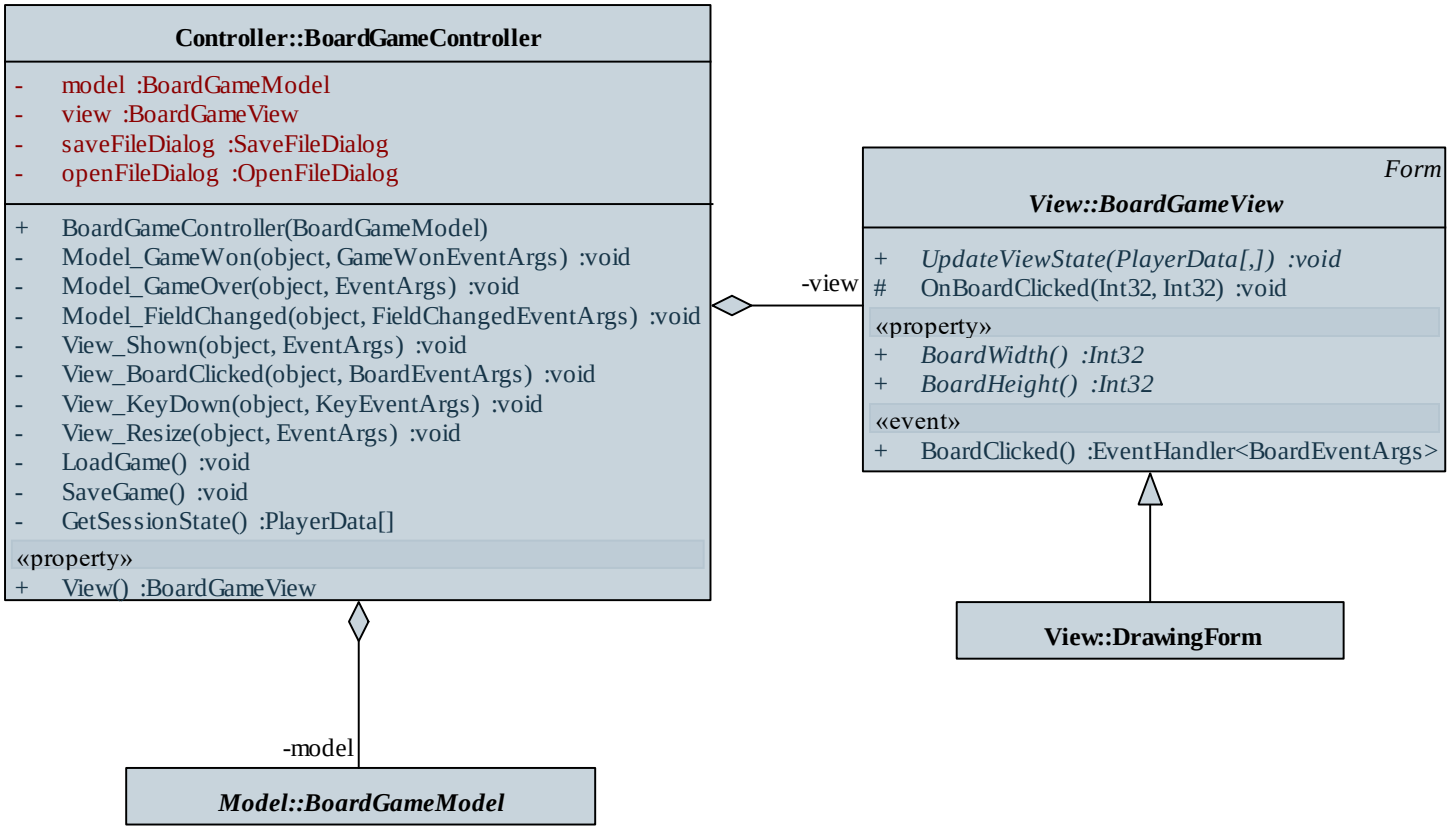
Feladat: Készítsünk egy táblajáték programot MVC architektúrában, amelyben két játékos küzdhet egymás ellen.

- a vezérlő (**`BoardGameController`**) látja a modellt, illetve létrehozza a nézetet, és kezeli a nézet eseményeit (billentyűzet lenyomás, átméretezés, táblakattintás)
- a nézet interfészét kibővítjük a táblával kapcsolatos információkkal (**`BoardWidth`**, **`BoardHeight`**)
- a táblakattintás (**`BoardClicked`**) egy összetett esemény, amelyet a nézetben váltunk ki
- betöltés alapú megközelítést alkalmazunk, így továbbra is a nézet feladata az állapotfrissítés (**`UpdateViewState`**) a kapott nézet modell alapján

Grafikus felületű alkalmazások architektúrái

Példa

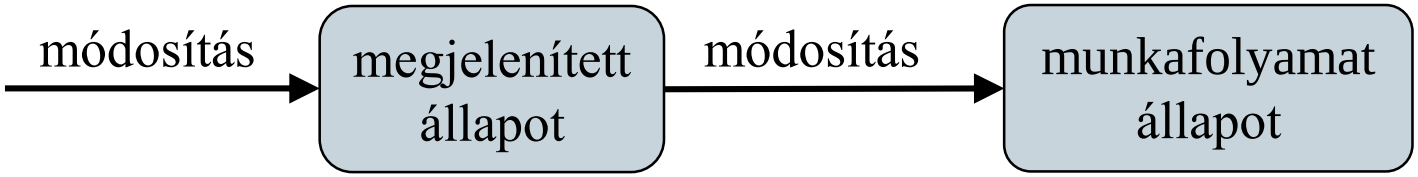
Tervezés:



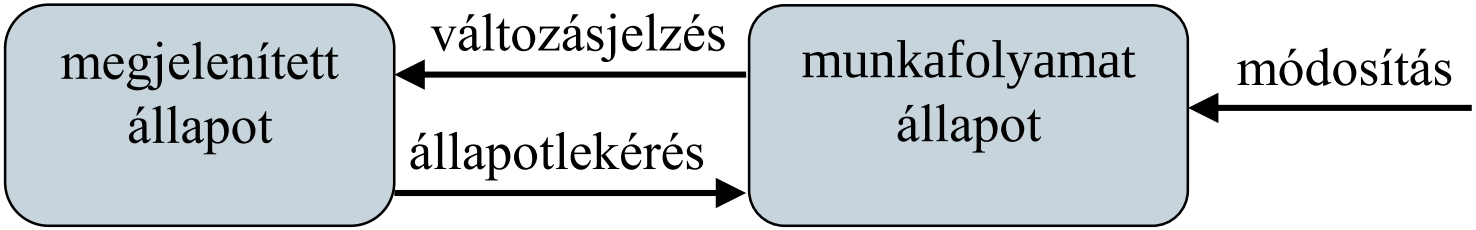
Grafikus felületű alkalmazások architektúrái

Adatkötés

- Az állapotok automatikus szinkronizálását *adatkötés* (*data binding*) segítségével érhetjük el, amely két lehetséges módon biztosíthatja a szinkronizációt
  - az érték módosítás hatására átíródhat a másik állapotba



  - az érték módosítás hatására jelzést adhat a másik állapot frissítésére



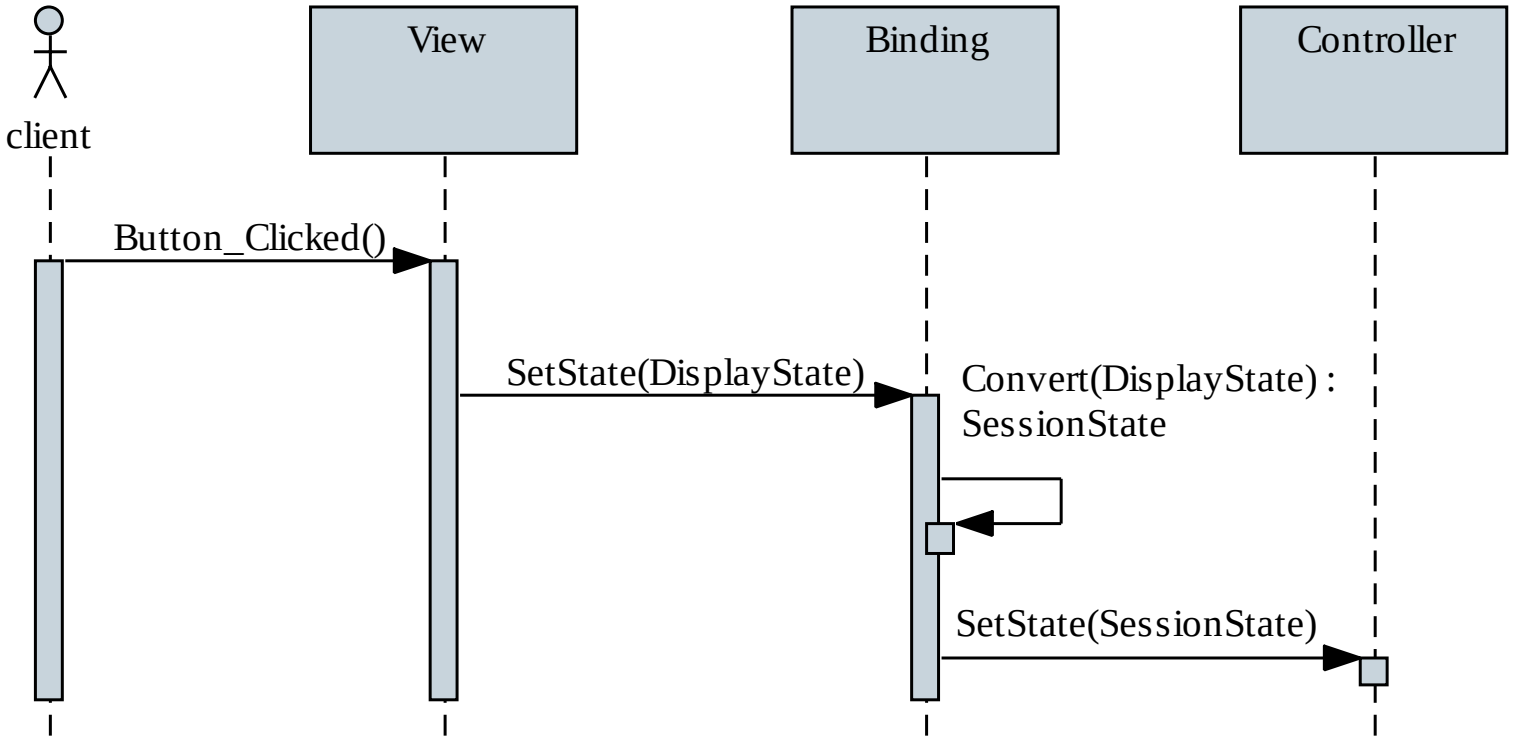
Grafikus felületű alkalmazások architektúrái

Adatkötés

- Az adatkötésért egy olyan adatkötés objektum (**`Binding`**) felel, amelyet a megjelenített állapot tárolója (nézet) és a munkafolyamat állapot tárolója (modell) közé helyezünk
  - az adatkötés ismeri mind a nézetet mind a modellt, ezért lehívhatja az értékeket (**`GetState`**), és elvégezheti a módosítást (**`SetState`**)
  - elvégezheti az átalakítást (**`Convert`**) a munkafolyamat állapot és a nézet állapot között
  - általában minden szinkronizálandó adathoz külön adatkötés tartozik, többszörös előfordulás esetén akár előfordulásonként is tartozhat adatkötés
  - a nézet ismerheti az adatkötést, így közvetlenül kezdeményezheti a módosítást az adatkötésen keresztül

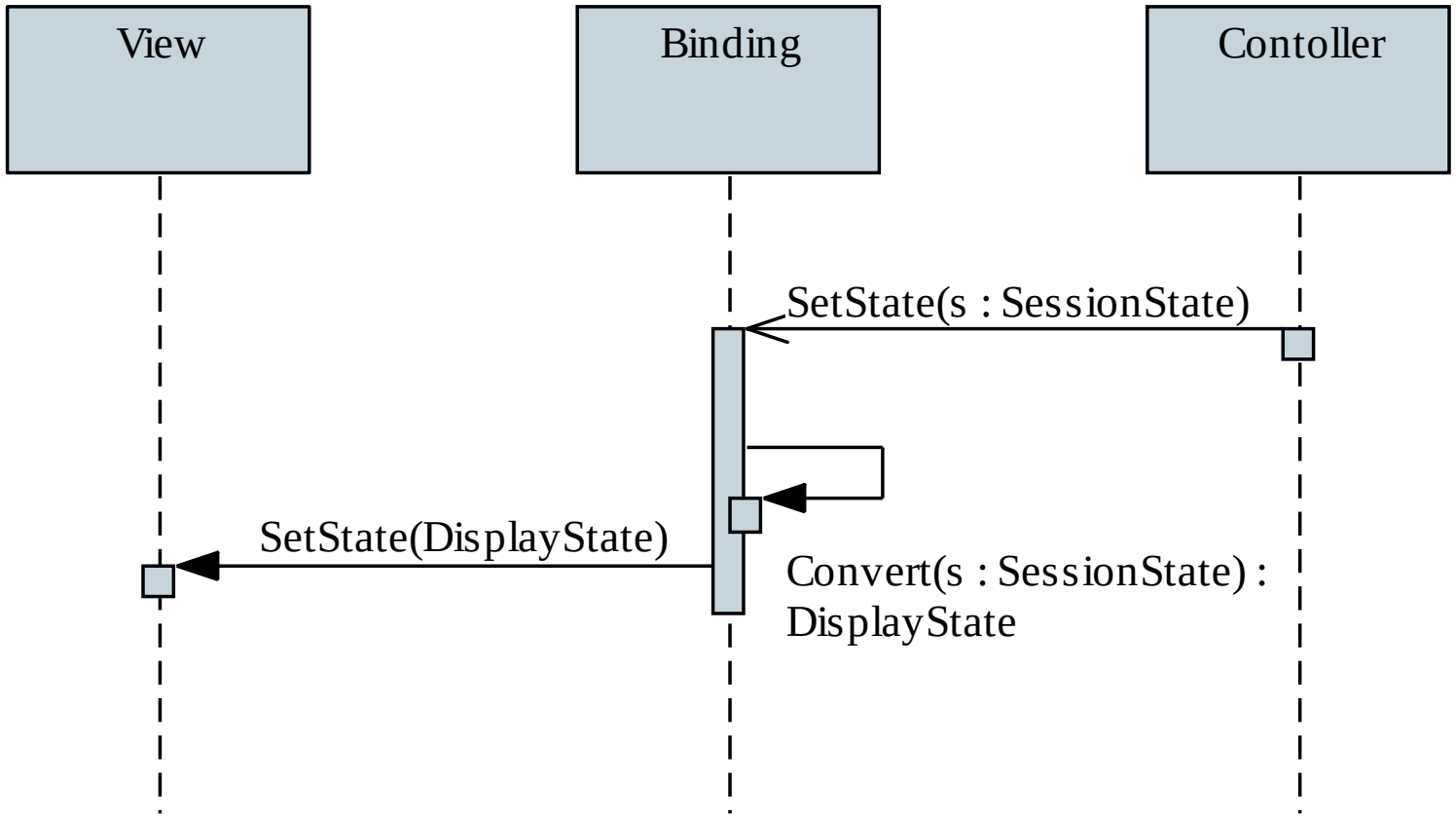
Grafikus felületű alkalmazások architektúrái

Adatkötés (modell irányába)



Grafikus felületű alkalmazások architektúrái

Adatkötés (nézet irányába)



Grafikus felületű alkalmazások architektúrái

Adatkötés megvalósítása

- Az adatkötés nyelvi támogatásként érhető el modern .NET platformokon (pl. WPF, WinRT, UWP, Xamarin)
  - a grafikus felületet deklaratívan, XAML segítségével írjuk le
  - a felületen a kötéseket a **`Binding`** típus segítségével adjuk meg a cél helyén, és megadjuk a forrást (**`Source`**), továbbá megadhatjuk az átalakítás módját (**`Converter`**), pl.:

```
<Label Text="{Binding Source=Value}" />
<!-- a feliratot adatkötéssel adjuk meg -->
```

  - magára a teljes felületre az adatforrást (a nézet modellt) a **`DataContext`** tulajdonság segítségével helyezhetjük, pl.:

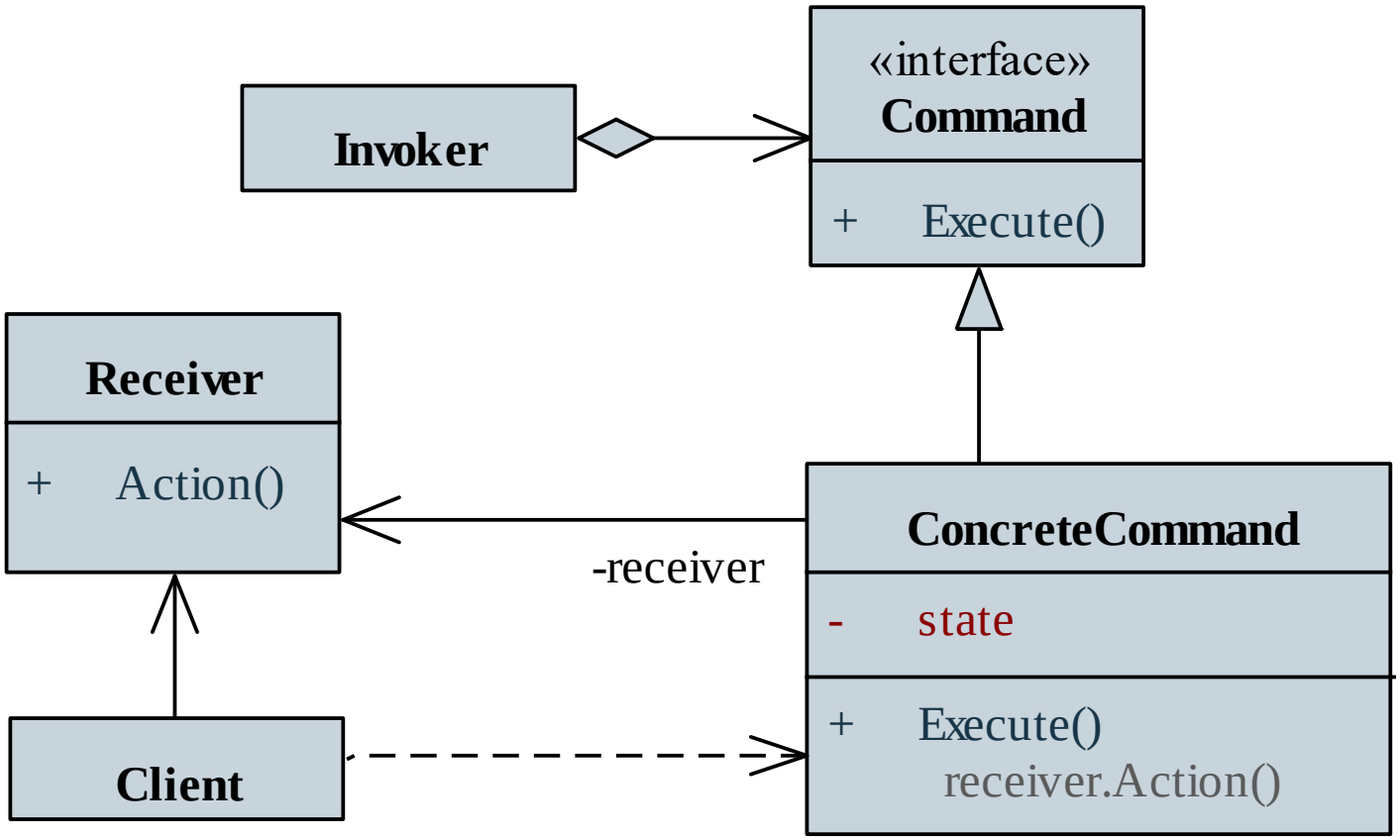
```
ViewModel vm = new ViewModel; // nézet modell
vm.Value = 0; // érték beállítása
view.DataContext = vm; // adatforrás megadása
```

Grafikus felületű alkalmazások architektúrái

Parancsok

- A *parancs* (*command*) tervminta célja egy művelet kiemelése egy külön objektumba, így azt több objektum többféleképpen is igénybe veheti
  - a végrehajtandó tevékenység (**`Action`**) formája, paraméterezése tetszőleges lehet, ezért nem lehet egyazon módon különböző környezetekben kezelni
  - a parancs (**`Command`**) egy egységes formát biztosít egy tevékenység végrehajtására (**`Execute`**), a konkrét parancs (**`ConcreteCommand`**) pedig biztosítja a megfelelő tevékenység végrehajtását
  - a kezdeményező (**`Invoker`**) csupán a parancsot látja, így a tényleges végrehajtás előle rejtett marad

Parancsok


«interface»
Command
+ Execute()
Invoker
Receiver
+ Action()
-receiver
ConcreteCommand
- state
+ Execute()
receiver.Action()
Client

Grafikus felületű alkalmazások architektúrái

Parancsok megvalósítása

- A parancsok nyelvi támogatásként érhetőek el modern .NET platformokon (pl. WPF, WinRT, UWP, Xamarin)
  - a parancs (**`ICommand`**) lehetőséget ad egy tetszőleges tevékenység végrehajtására (**`Execute`**), illetve a végrehajthatóság ellenőrzésére (**`CanExecute`**)
    - célszerű a tevékenységeket lambda-kifejezések (**`Action`**) formájában megvalósítani
  - a parancsokat adatkötés segítségével, a nézet modellen keresztül kapcsolhatjuk a grafikus felületre, ahol a vezérlők parancs (Command) tulajdonságához köthetjük, pl.:

```
<Button … Command="{Binding MyCommand}" />
<!-- a parancsot adatkötéssel adjuk meg -->
```

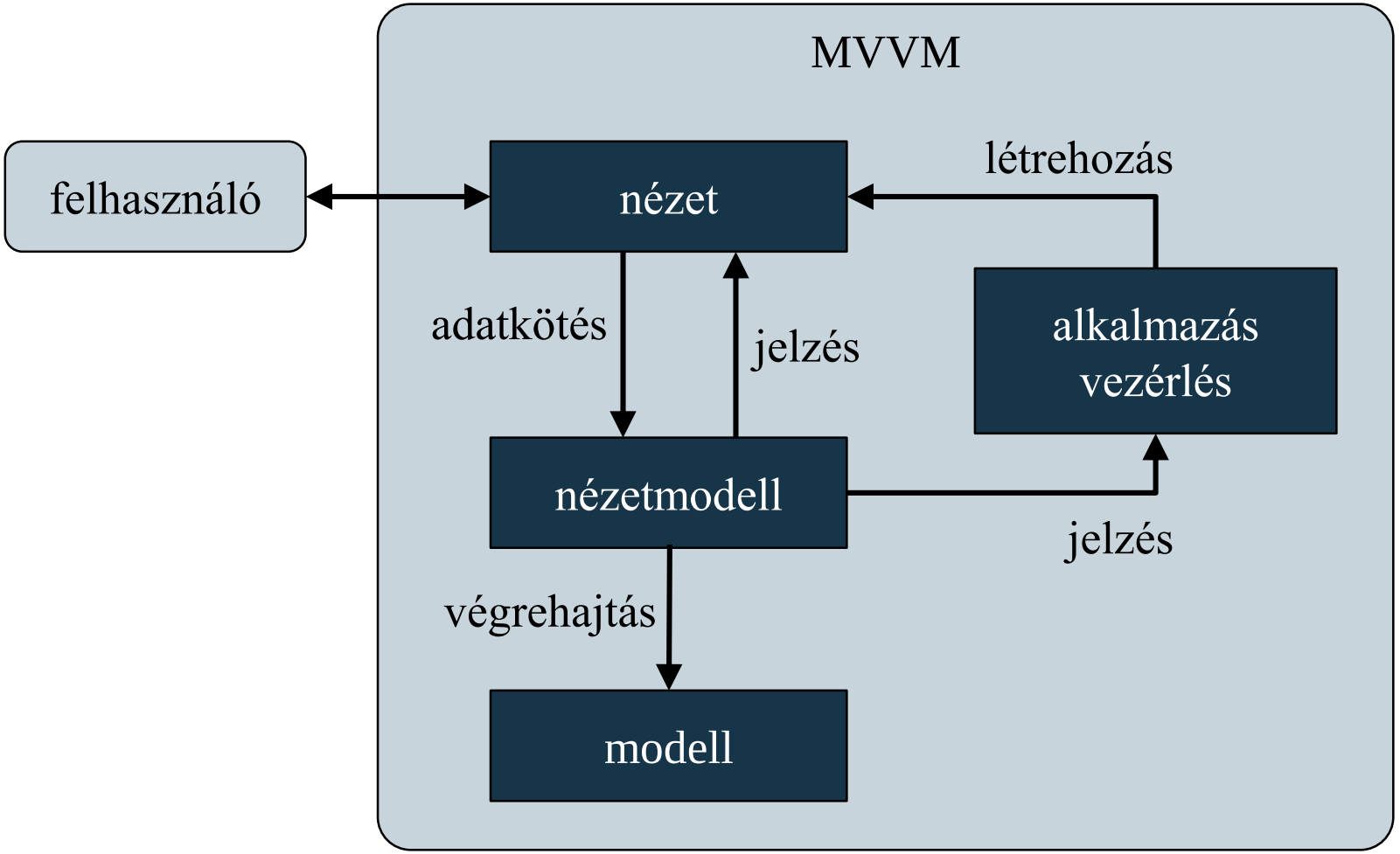
Grafikus felületű alkalmazások architektúrái

MVVM architektúra

- A *Model-View-Viewmodel* (*MVVM*), vagy *prezentációs modell* (*PM*) architektúra a nézettel kapcsolatos tevékenységeket, illetve a nézetállapot frissítését egy *nézetmodell* (*viewmodel*) komponensbe helyezi
  - a *nézetmodell* tartalmazza a felületi adatokat (megjelenített állapot), valamint a tevékenységek végrehajtásához szükséges műveleteket (*parancs*ok formájában)
  - a nézethez a megjelenített állapotot *adatkötés*sel kapcsoljuk, a nézetmodell pedig automatikusan jelez a nézetállapot megváltozásáról (*figyelő* segítségével)
- Az összetett tevékenységeket (pl. nézetek létrehozása) egy külön *alkalmazás vezérlés* (*application control*) biztosítja

Grafikus felületű alkalmazások architektúrái

MVVM architektúra



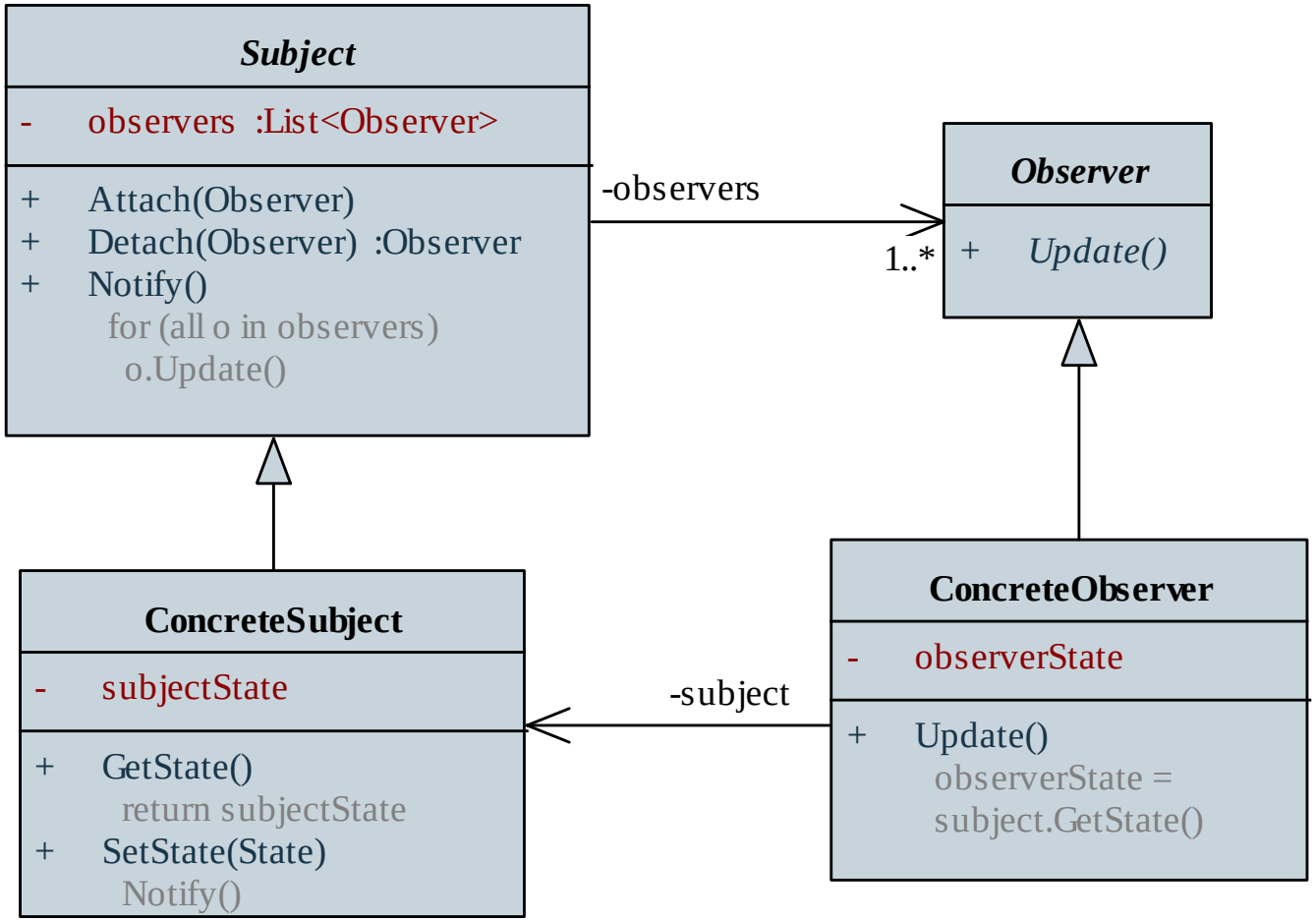
Grafikus felületű alkalmazások architektúrái

Figyelő

- A *figyelő* (*observer*) tervminta célja összefüggőség megadása az objektumok között, hogy egyik állapotváltozása esetén a többiek értesítve lesznek
  - a figyelő (**`Observer`**) ehhez biztosítja a változás jelzésének metódusát (**`Update`**)
  - a megfigyelt objektumok (**`Subject`**) az értékeikben tett változtatásokat jelzik a felügyelőknek (**`Notify`**)
  - egy objektumot több figyelő is nyomon kísérhet
  - az MVVM architektúrában a figyelő szerepét az adatkötés tölti be, míg a megfigyelt objektum a nézetmodell

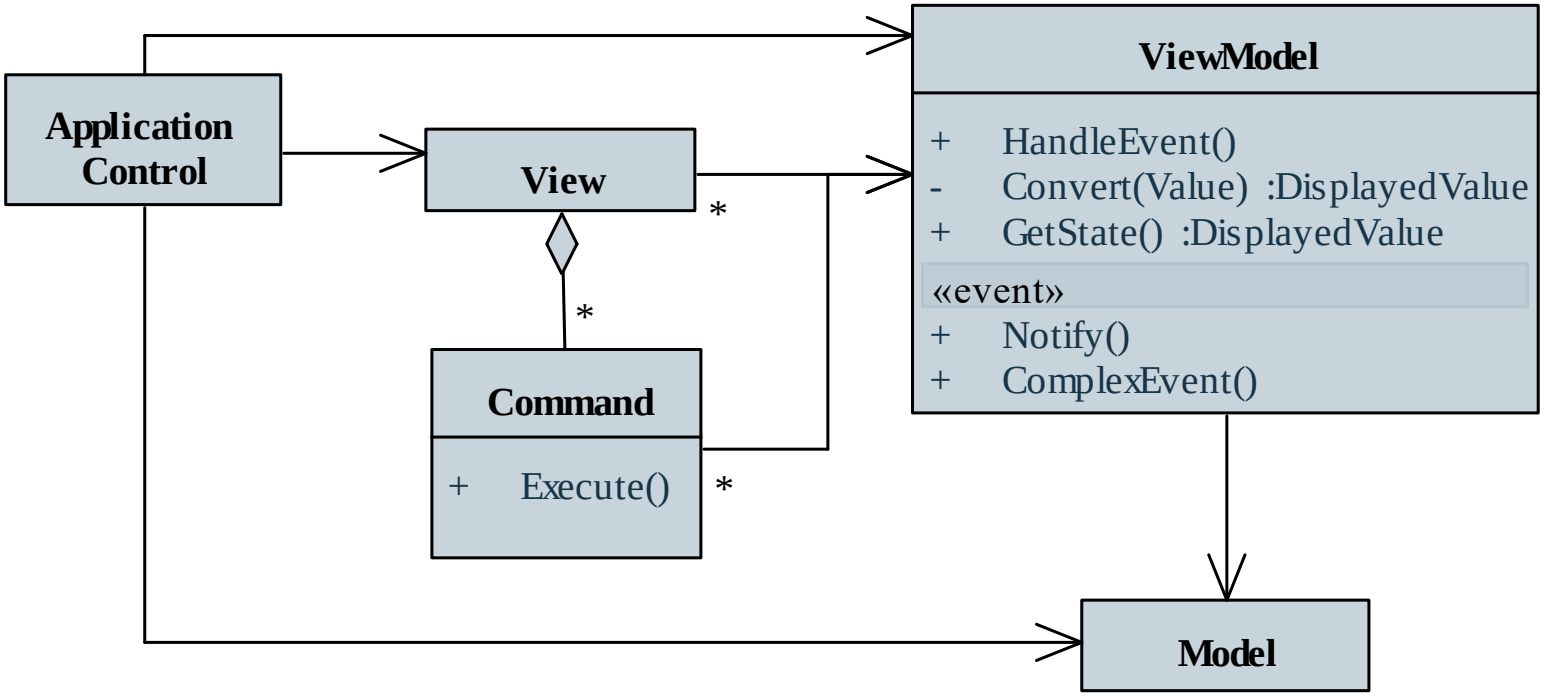
Grafikus felületű alkalmazások architektúrái

Figyelő



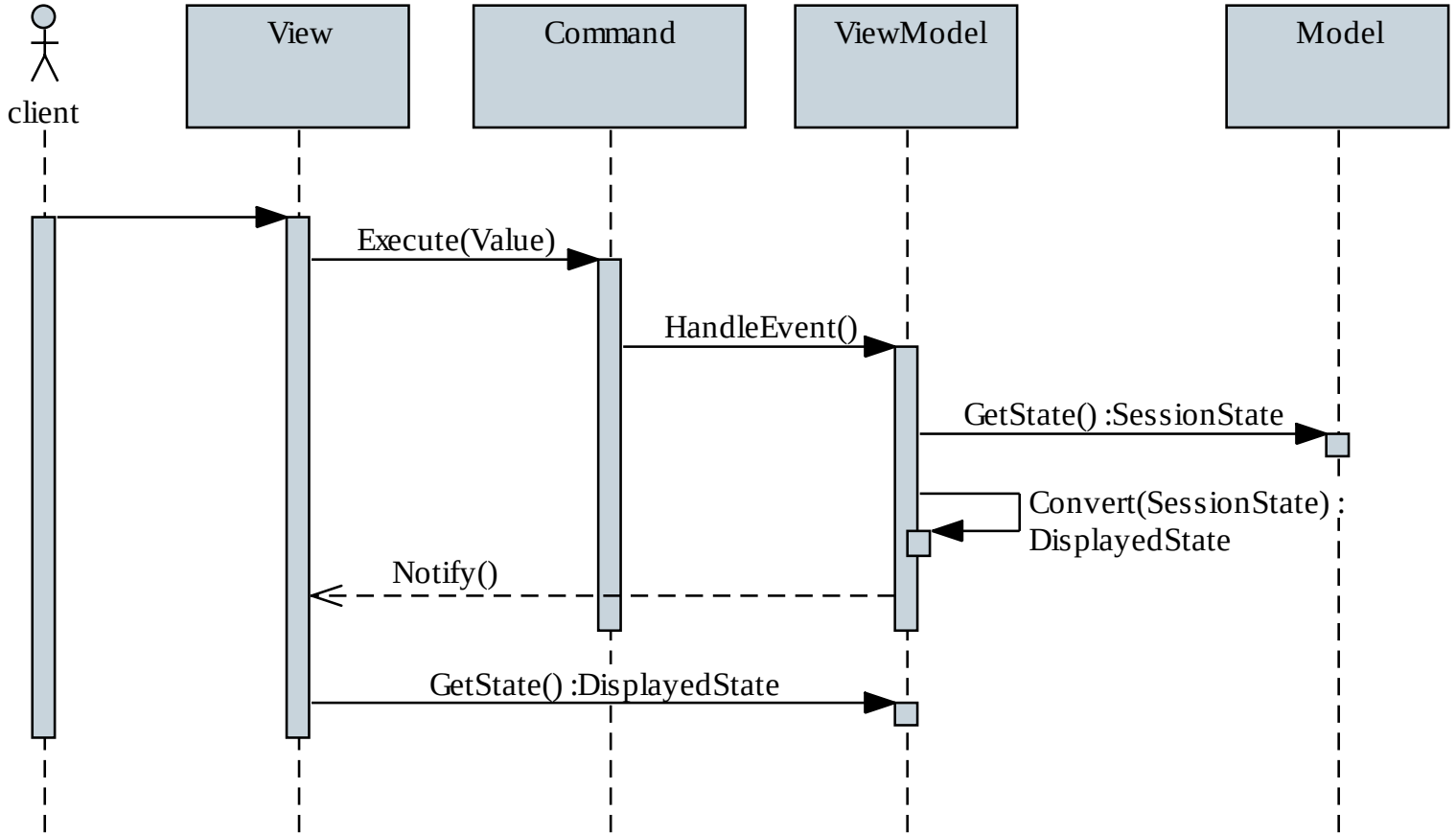
MVVM architektúra

- A nézetmodell tekinthető egy átjárónak (*proxy*), amely a nézet és a modell között helyezkedik el



Grafikus felületű alkalmazások architektúrái

MVVM architektúra



Grafikus felületű alkalmazások architektúrái

MVVM architektúra

- Előnyei:
  - a nézet csak a felület deklaratív leírását tartalmazza, minden tevékenység és adatkezelés (átalakítás) külön rétegben található
  - a felület teljesen függetlenül alakítható ki a nézetmodelltől
- Hátrányai:
  - összetett architektúra, alapos átgondolást, és sok beépített elemet igényel
  - a nézet és a nézetmodell összekötése közötti inkonzisztenciák csak futási időben derülnek ki

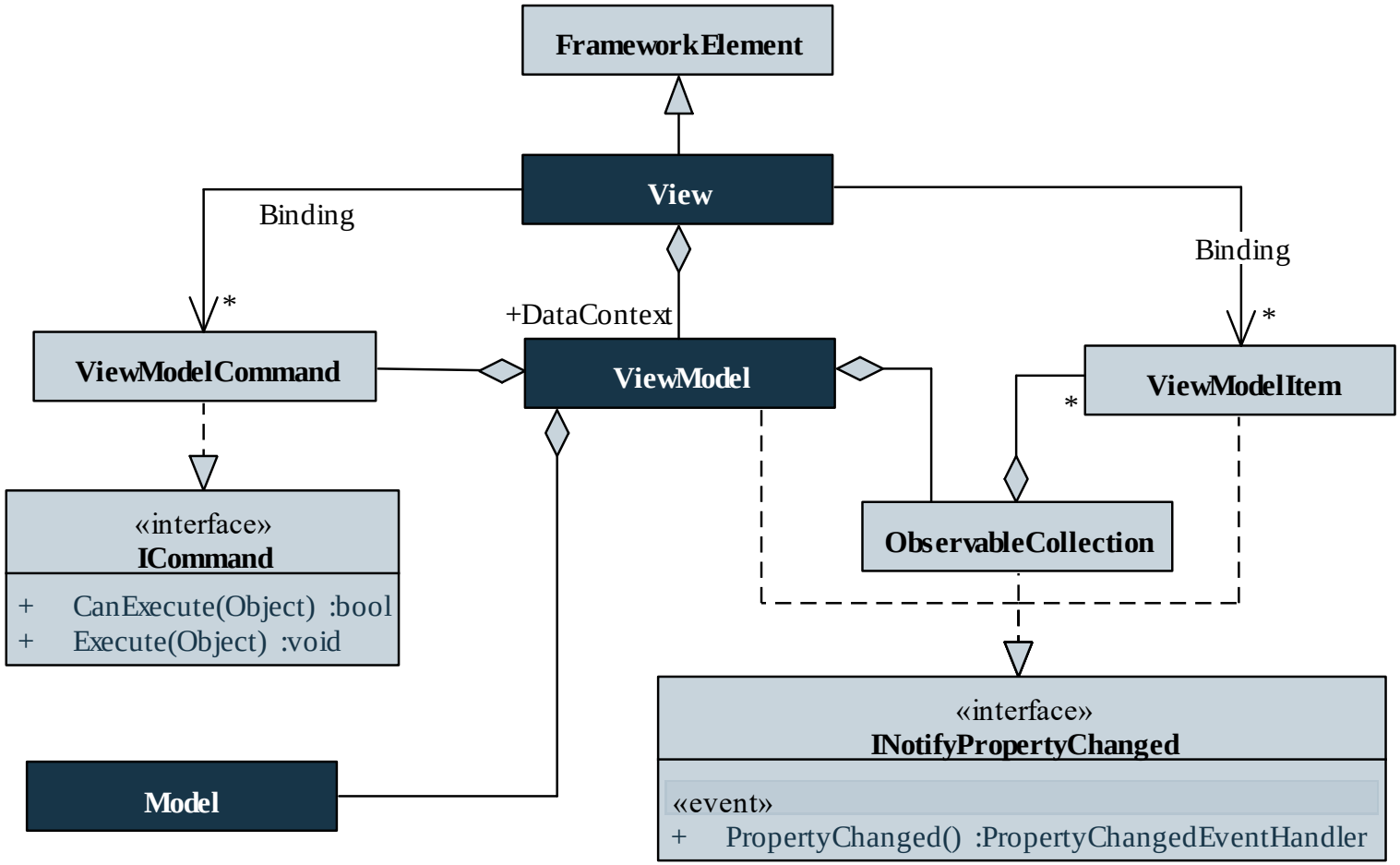
Grafikus felületű alkalmazások architektúrái

MVVM architektúra megvalósítása

- Az MVVM architektúra megvalósítása nyelvi támogatásként érhető el modern .NET platformokon (pl. WPF, WinRT, UWP, Xamarin)
  - használjunk az adatkötést (**`Binding`**) és a parancsokat (**`ICommand`**)
  - az adatokban (tulajdonságokban) történt változások nyomon követése a nézetmodellben történik (**`INotifyPropertyChanged`**)
    - a megadott tulajdonság módosításakor kiválthatjuk a **`NotifyPropertyChanged`** eseményt
  - gyűjteményekben bekövetkezett változásokat is nyomon követhetünk (**`INotifyCollectionChanged`**)

Grafikus felületű alkalmazások architektúrái

MVVM architektúra megvalósítása



Grafikus felületű alkalmazások architektúrái

Példa

Feladat: Készítsünk egy táblajáték programot MVVM architektúrában, amelyben két játékos küzdhet egymás ellen.

- külön projektet hozunk létre a nézetmodellnek (**`ViewModel`**), valamint a nézetnek (**`View.Presentation`**)
- kiegészítjük a nézetmodellt a változásjelzéssel (**`INotifyPropertyChanged`**), amelyet egy közös ősosztályban kezelünk, és váltunk ki (**`ViewModelBase`**)
- a tevékenységeket felbontjuk a nézetmodell és az alkalmazás (**`App`**) között
  - új nézetet igénylő tevékenységekről (pl. betöltés, mentés, játék vége) a nézetmodell, vagy a modell eseményt küld az alkalmazásnak

Grafikus felületű alkalmazások architektúrái

Példa

Tervezés: