**Capitolul 2. Analiza pattern-elor de proiectare**

**2.1 MVC(Model-View-Controller)**

Trygve Reenskaug a fost informatician norvegian care a adus mari contribuții în popularizarea POO (programării orientate pe obiecte) și a influiențat evoluția modelelor de dezvoltare a aplicațiilor moderne**.** Model-View-Controller a fost introdus ca concept de Trygve între anii 1970 și 1980 pentru a rezolva problema controlului a mai multor utilizator asupra unui set enorm și complex de date, modelul fiind implementat deja pentru limbaju de programare Smalltalk-80 [4].

**2.1.1 Arhitectura MVC**

MVC implementează separarea aplicația dezvoltată in trei compenente distincte: componenta de intrare, componenta de ieșire și cea de procesare a informației[3]. Modelul (Model) face legătura dintre aplicație și date stocate în memorie sau într-o bază de date. Vizualizarea (View) afișează datele în fața utilizatorului sau permite interacțiunea cu aplicația prin interfața de utilizator. Controlerul(Controller) deja descrie logica de transmitere a datelor de la model la vizualizare și executarea interacțiunilor cu aplicația.

Fiecare componenta poate interacționa cu celelalte, de exemplu o vizualizare afișează datele unui model, controlerul procesează o interacțiune cu utilizatorul de la o vizualizare sau modelul actualizează datele din bază de date transmise de controler, reprezentat în figura 2.1.1.1. În cel mai des caz vizualizarea și controlerul sunt luați ca o pereche din simplu motiv că au o interacțiune mai strânsă în modelul MVC[2]. Fiecare vizualizare are la baza un controler, iar fiecare controler este are setul său de vizualizări pe care le gestionează[2].

Acces la date

Notificări

Răspunsul primit la executare unei cerei

Chemarea unei funcționalități

Actualizarea datelor

Extragerea obiectului model

figura 2.1.1 Reprezentarea interacțiunilor între componentele MVC

**2.1.2 Modelele**

Modelul este un obiect ce conține toate datele folosite de aplicație de obice invizibil pentru utilizator în afara vizualizării prin care acesta interacționează. Pentru a ineracționa cu celelalte componente ale aplicației modele furnizează interfață de acces la datele stocate. Modelele deasemenea pot varia de la o simpla variabilă de tip intreg (int) pană la o clasă, cu toate beneficiile POO de la encapsularea datelor pentru protecția lor și posibilitatea de reutilizare a codului până la abstractizare, ce permite modelelor să interacționeze cu alte componente fără să știe de prezența acestora.[3] În alte cazuri ca modele sunt și serviciu complex cu mai multe nivele de arhitectura folosite pentru rezolvarea diverselor probleme sau implementarea diferitor funcționalități [3]. Cel mai frecvent pentru realiza legătura dintre date stocate și modele, care furnizează datele aplicației, sunt folosite ORM sau Object Relational Mapper, de exemplu EntityFramework pentru C#.

**2.1.3 Vizualizările**

Vizualizarea reprezintă prezentarea grafică a aplicației pentru utilizator, cu care acesta interacționează. Acestea permite acces la date prin interfață și interacționarea cu aceste date și însuși aplicația, dacă logica programului furnizează aceste opțiuni, prin diferite acțiuni ca click pe mouse sau tastatură. Vizualizările sunt de obicei realizate cu ajutorul limbajelor de markdown sau a vareațiilor acestora. Ca exemplu banal este html. Chiar dacă vizualizarea reprezintă datele sau acțiunile afectuate pe aceasta, ea nu știe de logica efectuarii acestor acțiuni [3]. View-urile sunt de asemenea imbricate de obicei având mai multe subview-uri in acestea[3].

**2.1.4 Controlere**

Controalele sunt fac de obicei legatura între vizualizări și modele și definește logica de transmitere a datelor dela una la alta într-un model MVC. Controlerul este responsabil de acceptarea input-ului utilizatorului de la vizualizare și executarea logicii programului conform datelor primite. Acest comportament implementează pattern-ul “Strategy” ce permite flexibiliatea și poate fi utilizat pentru a oferi un comportament editabil și unul needitabil în același formular.[3]

**2.2 MVVM(Model-View-ViewModel)**

În 2004, Marti Fowler (inginer softaware englez) a publicat un articol în care el a prezentat un pattern numit Presentation Model (PM). PM ca și MVP(Model-View-Presenter) separa vizualizarea de conportamentul și starea ei prin abstractizarea acesteia numită Presentation Model sau model de prezentare. Ca consecință vizualizarea devine o interpretare a modelului de prezentare [9].

Deja în 2005 John Gossman, unul dintre arhitecții framework-urilor WPF (Windows Presentation Foundation) și Silverlight ai companiei Microsoft, a prezentat pattern-ul MVVM sau Model-View-ViewModel în blog-ul său [9]. Ca și PM, MVVM avea la bază abstractizarea vizualizării în care conținută starea și comportamentul. Cu toate acestea MVVM a fost introdus ca o standartizare a modelului pentru a simplifica crearea interfețelor de utilizator prezentă în WPF, în timp ce PM se axa mai mult pe crearea acestor interfețe interfețelor.

**2.2.1 Arhitectura MVVM**

Arhitectura MVVM este destinată mai mult pentru crearea aplicațiilor cu interfețelor de utilizator moderne în care de vizualizare este mai responsabil un designer decât un dezvoltator clasic.[2]

Model-View-ViewModel este un pattern care include trei componente esențiale: Model, View și ViewModel, așa cum sugerează denumirea modelului. Ca și în cazul la MVC, vizualizarea și modelul au acela rol de a afișa datele sau interacționa cu utilizatorul și de a face legătura și validarea acestor date respectiv, însă în cazul la ViewModel sau model de vizualizare, acesta are ca menire să descrie logica, comportamentul și starea vizualizării, în afară de transmiterea datelor și prelucrarea lor. O simplă reptrezentare a interacțiunii între compenente este figura 2.2.1.

Actualizarea datelor

Acces la date

Chemarea unei acțiuni

Transmite rezultatul

figura 2.2.1 Reprezentarea interacțiunii între componentele MVVM

**2.2.2 Modelele**

Ca în cazul MVC-ului modelul are același rol de a face legătura cu datele din mediul de stocare și de a le furniza pentru afișare sau salva schimbările acestora. Acestea sunt de obicei clase model care incaplsulează datele. Exemple de modele includ repozitorii, business objects, obiecte de transfer de date (DTO), obiecte CLR simple (POCO) și obiecte entitate și proxy generate.

**2.2.2 Vizualizările**

Vizualizările sunt responsabile de definirea structurii, amplasării și aspectului elementelor care apar la ecran. În cele mai dese cazuri vizualizările sunt definite în XAML sau XML. În caz ideal vizualizările nu conțin în ele nici o logică de execuție ci doar referințe la funcții definite în model de vizualizare. O vizualizare poate avea propriul model de vizualizare sau poate moșteni modelul de vizualizare al părintelui său. O vizualizare primește date de la modelul său de vizualizare prin intermediul legăturilor sau al invocării metodelor din modelul de vizualizare[8]. În timpul execuției, vizualizarea se modifică atunci când controalele interfeței de utilizator primesc răspuns de la modelului de vizualizare prin execuția evenimentului legat de componenta vizualizării [8]. Din acțiuni care pot chema execuția unei metode pot fi de exemplu click, dublu click, etc.

**2.2.3 Modele de Vizualizări**

În cele mai dese cazuri modelele de vizualizare sunt descrise ca clase care fac legătura între model și vizualizare și sunt responsabile pentru definirea logicii de comportament anume a vizualizărilor. Cu alte cuvinte se poate de spus că viewmodel-ul interacționează cu modelul aplicației prin chemarea metodelor și extragerea datelor din clasa model și cu vizualizarea prin furnizarea acestor date și implementarea metodelor în formă de evenimente (events) ce descriu logica interacțiunii utilizatorului cu aplicația prin aceasta. Accesul modelului de vizualizare la starea vizualizării este efectuată prin “data binding” pentru a se sincroniza, ca să primească acces la elementele vizualizării și proprietăților acestora.

**2.3 MVP(Model-View-Presenter)**

Model-View-Presenter (MVP) a fost introdus pentru prima dată în sistemul de operare Taligent, companie formată din colaborarea Apple, IBM, și Motorola în 1996 [10]. Conceptul pattern-ului se bazează pe MVC cu definire mai clară a interacțiunii între componente. MVVM menționat mai devreve are ca predecesor MVP.

**2.3.1 Arhitectura MVP**

MVP a fost conceput pe baza a două întrebări: gestionarea datelor și interfața utilizator. Adică, cum să fie gestionate datele și cum să fie implementată interacționarea utilizatorilor cu datele[2]. Acest pattern de design a fost creat pentru a ușura dezvoltarea interfețelor grafice ca consecință a separării funcțiilor între vizualizare și componenta care implementează logica vizualizării. În formă simplificată modelul MVP conține trei componente: model, vizualizare și preseter ca și în cazul celorlalte pattern-e, însă în unele cazuri MVP este divizat în 6 componente: selections, commands, interactor și cele trei menționate mai devreme. În caz general reprezentarea componentelor și interacțiunilor între acestea este în figura 2.3.1

Actualizarea vizualizării

Actualizarea datelor

Acces la date

Chemarea unei acțiuni

figura 2.3.1 Reprezentarea interacțiunii între componentele MVP

**2.3.2 Modele**

Din cauză că MVP este succesorul MVC și predecesorul MVVM, ca în celelalte cazuri modelele au aceeași structură și funcție în cadrul aplicației.

**2.3.2 Vizualizările**

Vizualizările iarăși au aceeași funcție ca și în MVC și MVVM cu excepția faptului că acestea nu au acces direct la model, ci cominică cu modelul prin prezentatoare avidențiind prin aceasta diferența față de predecesorul MVC. În cazul MVP vizualizările sunt definite ca interfețe care sunt implementate de preseter cu componentele acesteia fiind definite în limbajelor de markup ceea ce permite preseterului să manipuleze cu vizualizarea direct.

**2.3.3 Prezentatoare**

Prezentatorul este o dezvoltare a controlerului din Model-View-Controller și are ca menire implementarea logicii funcționării aplicației prin efectuarea legăturii între model și vizualizare. Spre deosebire de echivalentul său în MVC prezentatorul decide interacțiunea cu vizualizarea și are acces complet aceasta și elementele acesteia cât și parametrii acestora.

Bibiliografie:

1. [iamprovidence, MVC vs MVP vs MVVM with C# examples, https://medium.com, 2022](<https://medium.com/@iamprovidence/mvc-vs-mvp-vs-mvvm-with-c-examples-8013745e3c4c>)

2. [Tian Lou, A Comparison of Android Native App Architecture – MVC, MVP and MVVM, Master’s Thesis, Espoo, September 06, 2016](<https://pure.tue.nl/ws/portalfiles/portal/48628529/Lou_2016.pdf>)

3. [[Model-View-Controller architectural pattern and its evolution in graphical user interface frameworks](https://lutpub.lut.fi/handle/10024/92156)]<https://lutpub.lut.fi/bitstream/handle/10024/92156/Model-View-Controller%20architectural%20pattern%20and%20its%20evolution%20in%20graphical%20user%20interface%20frameworks.pdf?sequence=2&isAllowed=y>

4. [The original MVC reports Trygve Reenskaug Dept. of Informatics University of Oslo](<https://folk.universitetetioslo.no/trygver/2007/MVC_Originals.pdf>)

5. [Architectural Design of Modern Web Applications](<https://www.researchgate.net/profile/Lech-Madeyski/publication/221679095_Architectural_Design_of_Modern_Web_Applications/links/5c8fdf3292851c1df94a5233/Architectural-Design-of-Modern-Web-Applications.pdf>)

6. [MVVM Revisited: Exploring Design Variants of the Model-View-ViewModel Pattern] https://conferences.big.tuwien.ac.at/biweek2024/pdfs/biweek2024\_paper\_87.pdf

7. [The MVVM Pattern] <https://conferences.big.tuwien.ac.at/biweek2024/pdfs/biweek2024_paper_87.pdf>

8. [Model-View-ViewModel (MVVM)]<https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm>

9. [Patterns - WPF Apps With The Model-View-ViewModel Design Pattern] <https://learn.microsoft.com/en-us/archive/msdn-magazine/2009/february/patterns-wpf-apps-with-the-model-view-viewmodel-design-pattern>

10. [MVP: Model-View-Presenter The Taligent Programming Model for C++ and Java] <https://www.wildcrest.com/Potel/Portfolio/mvp.pdf>

11. [Model View Presenter] https://anshul-vyas380.medium.com/model-view-presenter-b7ece803203c