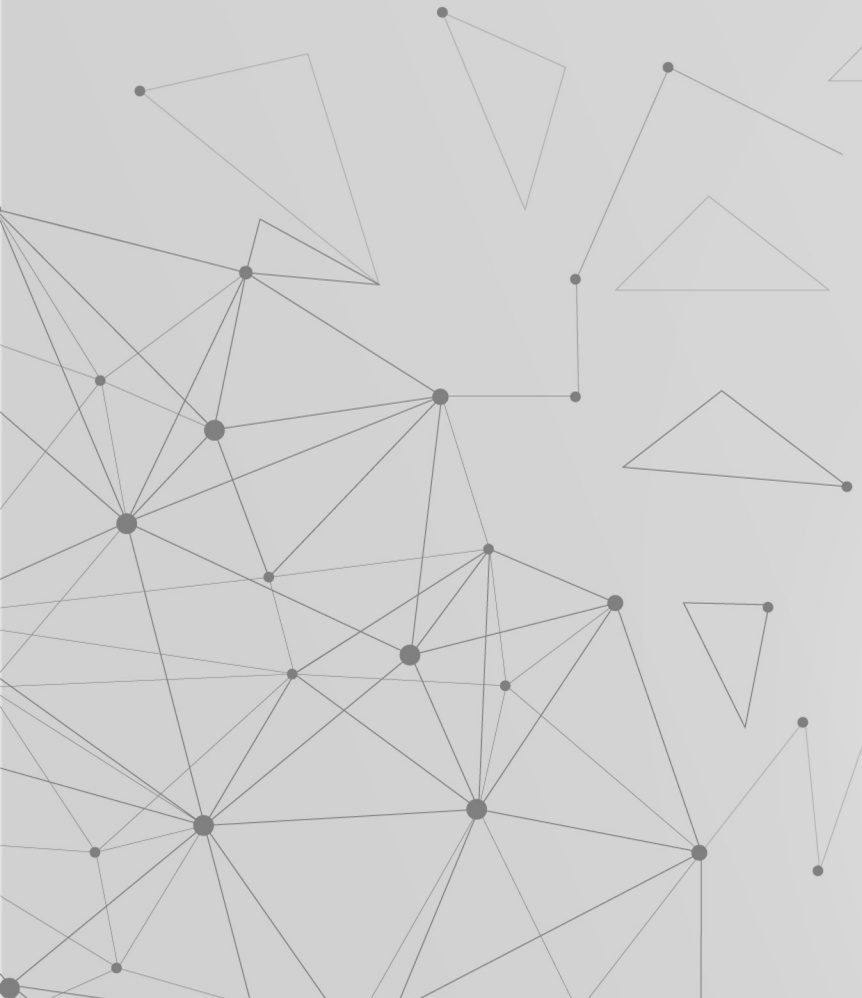


Pattern of The Week

Week 4: Mediator Pattern & CQRS

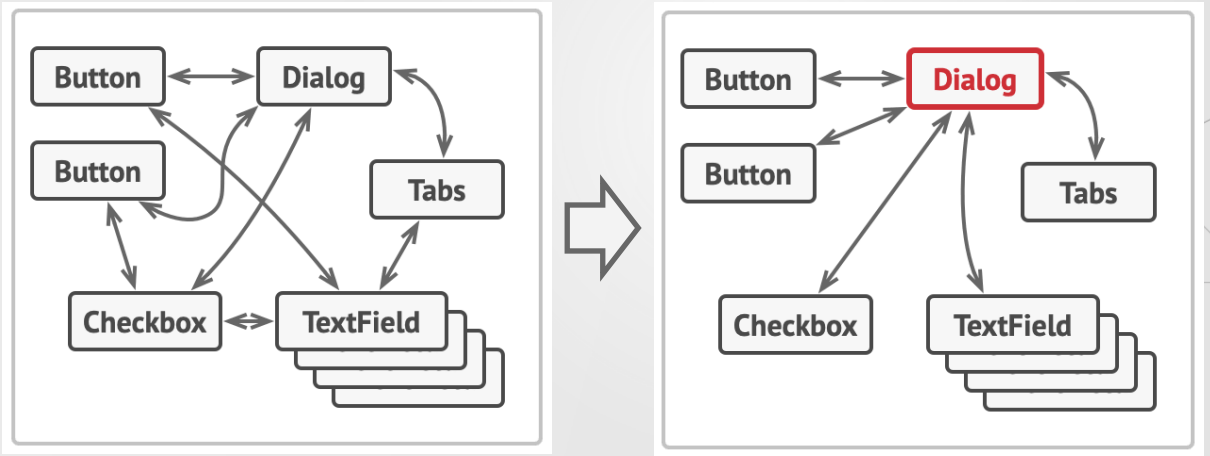
TABLE OF CONTENTS

01

Mediator Pattern

Mediator Pattern

- Reduzieren der Kopplung
- Mediator als Vermittler zwischen den Komponenten



Das original UML

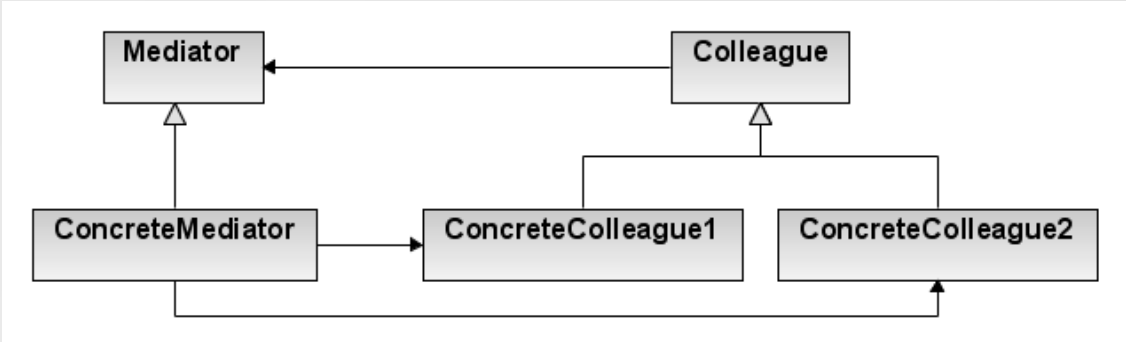
```
interface IComponent
{
    void SetState(object state);
}

class Component1 : IComponent
{
    internal void SetState(object state)
    {
        throw new NotImplementedException();
    }
}

class Component2 : IComponent
{
    internal void SetState(object state)
    {
        throw new NotImplementedException();
    }
}

// Mediates the common tasks
class Mediator
{
    internal IComponent Component1 { get; set; }
    internal IComponent Component2 { get; set; }

    internal void ChangeState(object state)
    {
        this.Component1.SetState(state);
        this.Component2.SetState(state);
    }
}
```



- ComponentX kennt den IMediator (Constructor Injection)
- Ähnlich wie Observer-Pattern (Broadcast)
- Der Mediator weiß, welche Komponente aufgerufen wird
- Die Komponenten kennen sich nicht

Aufgabe

Mediator implementieren

Neues Projekt

IComponent (SetState)

ComponentA (DoA), ComponentB (GetB)

IMediator, Mediator mit

SetState(), DoA(), GetB()

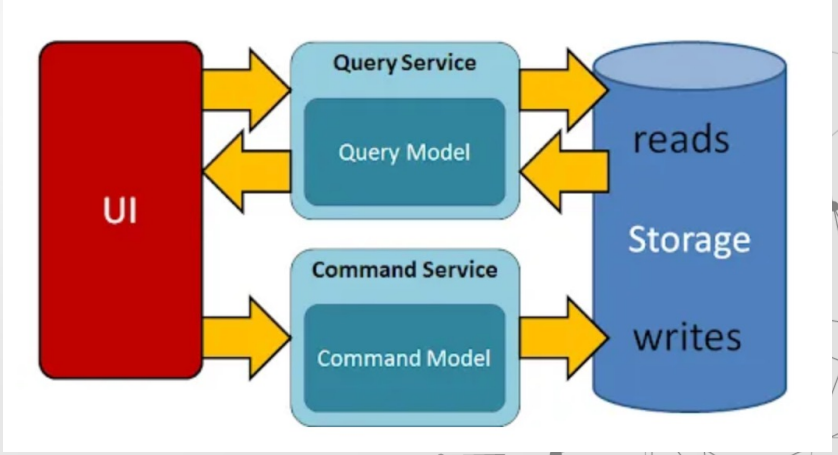
02

CQRS

Command and Query Responsibility Segregation

Command and Query Responsibility Segregation

- Trennung von Command (C*UD) und Query (*R**)
- Entkoppeln und ”Separation of Concerns”
- Jedes der C-R-U-D bekommt einen eigenen Service (Klasse)



Aufgabe/Demo

CQRS implementieren

Mediator Projekt

CQRS implementieren

Components werden Services

Parameter werden Commands oder Queries

03

MediatR

Nachteile des Mediator Pattern

- Mediator als “Dispatcher”
- Kennt alles, sozusagen “God-Object”

- Lose Kopplung aus Sicht der Komponenten
- Starke Kopplung aus Sicht des Mediators

→ Wie kann der Mediator wirlich ein “Mediator” sein?

MediatR

- Simple mediator implementation in .NET
- In-process messaging with no dependencies.
- Supports
    a. request/response,
    b. commands,
    c. queries,
    d. notifications and events,
- synchronous and async with intelligent dispatching via C# generic variance.

https://github.com/jbogard/MediatR

Mediator ➔ MediatR

- Gedanklicher Schritt von C++ nach C# (Generics und Reflection)
- Rückgabewerte werden Klassen (z.B. `ResponseX` Klasse)
- Methodensignaturen werden zu Klassen

    z.B. `RequestX` Klasse
- Generisches Interface werden Implementiert

    z.B. `IRequestHandler<RequestX, ResponseX>`
- Mediator hat nur noch die Methode “Send<T>(T request)”

Aufgabe

MediatR implementieren

Neues Projekt

ServiceCollection von .net 8

MediatR Bibliothek mit DI

Request, Validate, Notification