Patrón Mediator

Sistema de Chat Grupal - Análisis e Implementación

Universidad | Arquitectura de Software
Taller de Patrones de Diseño - Ejercicio 3

Tabla de Contenidos

1. Introducción

Este documento presenta el análisis, diseño e implementación de un **sistema de chat grupal** utilizando el **Patrón Mediator** de diseño. El proyecto demuestra cómo este patrón arquitectónico resuelve eficazmente los problemas de comunicación compleja entre múltiples objetos en un entorno de chat colaborativo.

> **Objetivo Principal:** Demostrar la aplicación práctica del Patrón Mediator para resolver problemas de alta dependencia y acoplamiento en sistemas de comunicación multi-objeto.

2. Análisis del Escenario

2.1 Problema Original

En un sistema de chat grupal tradicional, cada usuario necesitaría mantener referencias directas a todos los demás participantes, generando una red compleja de dependencias que presenta múltiples desafíos:

Problemas Identificados:

- **Alta dependencia:** Cada usuario debe conocer a todos los demás
- **Dificultad de escalabilidad:** Agregar/eliminar usuarios requiere modificar múltiples clases
- **Lógica dispersa:** La gestión de comunicación está distribuida en cada usuario
- **Mantenimiento complejo:** Cambios en la comunicación afectan múltiples componentes

2.2 Complejidad sin Mediator

En un sistema sin mediador, la complejidad de las relaciones crece exponencialmente:

Estructura Problemática (Sin Mediator)

Usuario A ↔ Usuario B ↔ Usuario C ↔ Usuario D

Complejidad: $O(n^2)$ dependencias

3. Justificación del Patrón Mediator

3.1 Comparación con Otros Patrones

Patrón	Ventajas	Desventajas para este escenario
Observer	Desacoplamiento parcial	No centraliza la lógica de comunicación
Command	Encapsula operaciones	No resuelve la comunicación entre objetos
Facade	Simplifica interfaz	No maneja interacciones complejas
Mediator	☑ Centraliza comunicación ☑ Reduce acoplamiento ☑ Facilita escalabilidad	Complejidad inicial

3.2 Razones de Elección del Mediator

🎯

Centralización

Toda la lógica de comunicación está en un solo lugar

🔗

Desacoplamiento

Los usuarios no conocen entre sí, solo al mediador

Escalabilidad

Agregar/eliminar usuarios es trivial

Mantenibilidad

Cambios en la comunicación se realizan en un solo punto

4. Implementación de la Solución

4.1 Estructura de Clases

```
// Interfaz del Mediador public interface IChatMediator { void SendMessage(string from, string to, string message); void AddUser(User user); } // Implementación Básica internal class ChatMediator : IChatMediator { private readonly IDictionary<string, User> _users; // Implementación de la lógica de comunicación } // Implementación Avanzada con Reflexión public class ReflectiveChatMediator : IChatMediator { private readonly IDictionary<string, User> _users; private readonly IDictionary<string, MethodInfo> _messageHandlers; private readonly IDictionary<string, object> _participants; // Implementación con capacidades reflexivas }
```

4.2 Participantes del Patrón

1. **Mediator (IChatMediator):** Define la interfaz para comunicación
2. **ConcreteMediator (ChatMediator/ReflectiveChatMediator):** Implementa la lógica de comunicación
3. **Colleague (User):** Representa los usuarios del chat
4. **ConcreteColleague (User):** Implementación específica de usuario

5. Arquitectura del Sistema

5.1 Flujo de Comunicación

Flujo Simplificado con Mediator

Usuario 1 → Mediador → Usuario 2

Usuario 1 → Mediador → Usuario 3

Usuario 1 → Mediador → Usuario 4

Complejidad: O(n) dependencias

5.2 Componentes Principales

ReflectiveChatMediator - Implementación Avanzada

- **Reflexión:** Uso de atributos para configuración automática
- **Handlers dinámicos:** Métodos marcados con `[MediatorHandler]`
- **Estadísticas:** Recopilación automática de métricas
- **Logging:** Registro de actividades del chat

User - Participante del Chat

```
[MediatorParticipant("ChatUser")] public class User { [MediatorProperty("NickName", true)] public string NickName { get; set; } [MediatorProperty("IsOnline")] public bool IsOnline { get; set; } // Métodos de comunicación a través del mediador }
```

6. Beneficios Obtenidos

6.1 Beneficios Esperados (Cumplidos)

☑ 1. Facilita el Mantenimiento

Antes: Modificar comunicación requería cambios en múltiples clases

Después: Cambios centralizados en el mediador

Ejemplo: Agregar filtro de mensajes solo requiere modificar el mediador

☑ 2. Mejor Organización

Antes: Lógica de comunicación dispersa en cada usuario

Después: Lógica centralizada y bien estructurada

Ejemplo: Manejo de usuarios offline, logs, estadísticas en un solo lugar

☑ 3. Reduce la Complejidad

Antes: Red compleja de dependencias entre usuarios

Después: Estructura simple: Usuario ↔ Mediador ↔ Usuario

Ejemplo: Agregar 10 usuarios nuevos no requiere modificar usuarios existentes

6.2 Beneficios Adicionales

Aspecto	Sin Mediator	Con Mediator
Acoplamiento	Alto ($O(n^2)$)	Bajo ($O(n)$)
Mantenimiento	Difícil	Fácil
Escalabilidad	Limitada	Excelente

Aspecto	Sin Mediator	Con Mediator
Testabilidad	Compleja	Simple

---

7. Características Avanzadas

7.1 Sistema de Reflexión

```
// Configuración automática de propiedades private void ConfigureParticipantProperties(object participant) { var properties = participant.GetType().GetProperties() .Where(p => p.GetCustomAttribute<MediatorPropertyAttribute>() != null); // Configuración automática basada en atributos }
```

7.2 Handlers Dinámicos

```
[MediatorHandler("broadcast", "Maneja mensajes enviados a todos los usuarios")] private void HandleBroadcastMessage(string from, string to, string message) { // Lógica específica para mensajes de difusión }
```

7.3 Estadísticas y Monitoreo

```
public Dictionary<string, object> GetChatStatistics() { return new Dictionary<string, object> { ["TotalUsers"] = _users.Count, ["TotalHandlers"] = _messageHandlers.Count, ["LastActivity"] = DateTime.Now }; }
```

8. Conclusiones

Resumen de la Implementación

El **Patrón Mediator** resultó ser la solución óptima para este escenario de chat grupal porque:

1. **Resuelve el problema central:** Elimina las dependencias directas entre usuarios
2. **Centraliza la lógica:** Toda la comunicación pasa por un punto único
3. **Facilita la escalabilidad:** Agregar/eliminar usuarios es trivial
4. **Mejora el mantenimiento:** Cambios se realizan en un solo lugar
5. **Permite extensibilidad:** Nuevas funcionalidades sin afectar usuarios existentes

Innovaciones Implementadas

- **Mediador Reflexivo:** Uso de atributos y reflexión para configuración automática
- **Handlers Dinámicos:** Sistema de procesamiento de mensajes basado en atributos
- **Estadísticas Integradas:** Monitoreo automático de la actividad del chat
- **Gestión de Estado:** Control avanzado del estado de los participantes

Impacto en la Calidad del Código

- **Acoplamiento:** Reducido de alto a bajo
- **Cohesión:** Aumentada significativamente
- **Mantenibilidad:** Mejorada considerablemente
- **Escalabilidad:** Excelente para crecimiento futuro
- **Testabilidad:** Cada componente es independiente
