

SWD Pattern

Mediator Pattern – Gruppe 13



16. april 2018

|  |  |
| --- | --- |
| **Navn:** | **Studienr:** |
| **Frederik Kastrup Mortensen** | **201607221** |
| **Stefanie Nielson** | **201605114** |
| **Søren Schou Mathiasen** | **201605264** |
| **Alexander Lichtenstein Davidsen** | **201608479** |

Indhold

[Introduktion 2](#_Toc511218619)

[UML Arkitektur 2](#_Toc511218620)

[Sammenligning 2](#_Toc511218621)

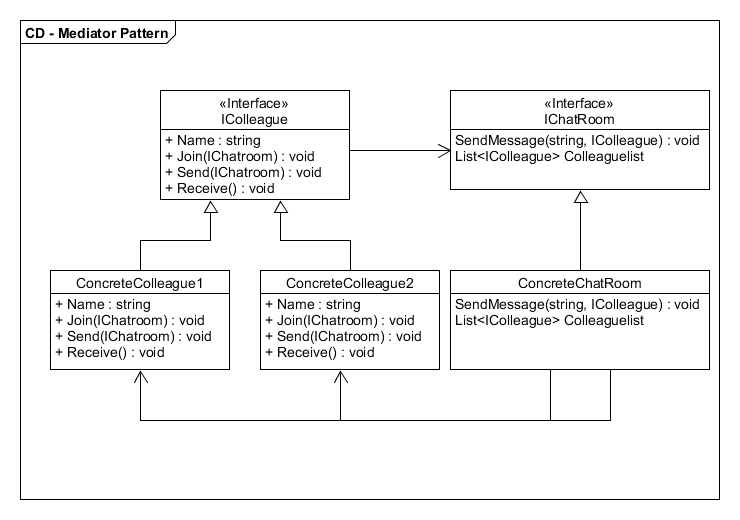
[Implementering 2](#_Toc511218622)

[Konklusion 2](#_Toc511218623)

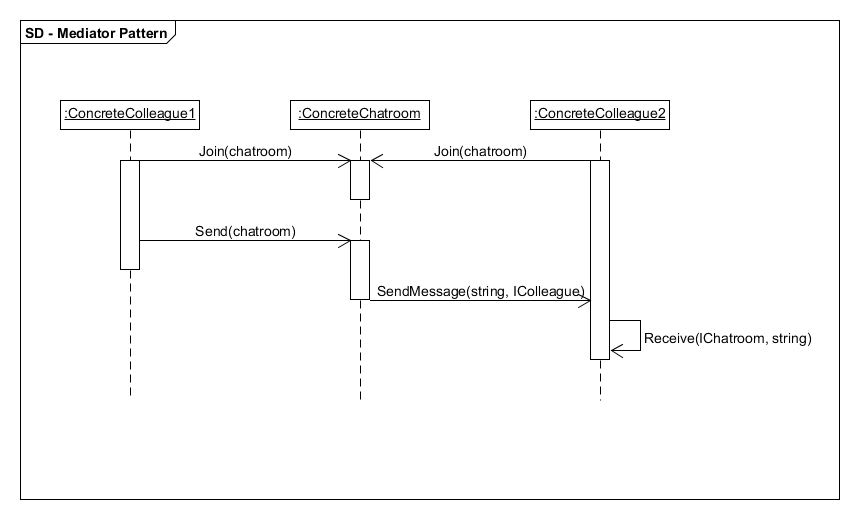
# Introduktion

Mediator pattern er et behavioral pattern. Formålet ved pattern er at det skal ændre måden objekter kommunikere med hinanden på, ved at tilføje et ekstra lag. Objekter kommunikerer gennem en mediator, så objekter ikke kommunikerer direkte med hinanden. Dette reducerer koblingen mellem de forskellige klasser. Mediator sørger for at de rigtige objekter får de rigtige beskeder. Dette bliver smart når et program opnår en stor mængde klasser med mange objekter der skal kommunikere på kryds og tværs. Dette gør programmet svære at læse og vedligeholde.

# UML Arkitektur

Klassediagram over media pattern. 

Sekvensdiagram over mediator pattern. De to Colleague objekter kalder join funktion på samme chatroom. Colleague 1 sender en besked til chatroom. Chatrrom sender herefter beskeden videre til Colleague 2 og colleague 2 kalde receive for at modtage beskeden.



# Sammenligning

**Mediator VS publisher/subscriber**

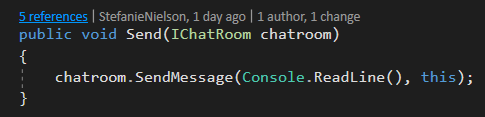
Hovedessensen i mediator-pattern er at man har en mediator som objekter kan subscribe til. Det er en slags ”posthus” hvor al posten bliver indleveret, og denne så står for at uddele posten til de respektive modtagere.   
Dette skaber en lav kobling i og med at afsendere og modtagere ikke behøver at kende til hinanden, men blot behøver at kende til denne mediator.

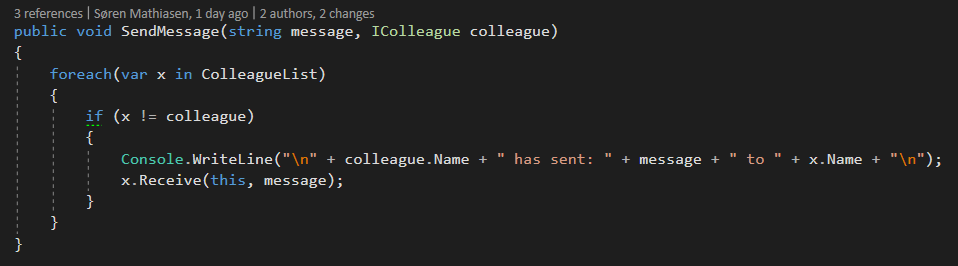
publisher-subscriber pattern går ud på at objekter subscriber sig ind på en given publisher. Det kunne f.eks. være at nogle objekter kun ønskede at vide noget om de målte sensorværdier i et system. Så ville de subscribe sig ind på den publisher der afsendte beskeder om dette. Når publisheren afsender en besked, vil kun de der har subscribet, modtage beskeden.

I og med at begge patterns går ud på at objekter kan subscribe på den ”publisher” de ønsker modtaget beskeder fra, er det også meget nærliggende at de to patterns kommer til at ligne hinanden rigtig meget.

Dette har det også gjort vi vores tilfælde.   
Hvis man tog eksemplet med posthuset igen, ville én af hovedforskellene være, at der i mediator-pattern bliver sørget for at den respektive modtager også modtager beskeden, hvor at man i publisher-subscriber pattern derimod blot sørger for at der bliver publishet en besked, og hvis der så er subscribers, vil de modtage beskeden, og ellers ikke.

Vi har valgt at opbygge vores system som et chatroom, hvor chatroomet er mediatoren.   
Dette betyder at de der afsender beskeden, vil indsende beskeden til chatroomet, som skal sørge for at ”uddele” disse beskeder.   
Dette ses nedenfor hvor colleaguen’s funktion Send(), kalder chatrooms funktion SendMessage():



Chatrooms SendMessage() skriver så denne besked til alle colleagues i chatroomet, undtagen den colleague der har skrevet beskeden.

Da systemet er opbygget således at det er et ”offentligt” chatroom – Altså at beskederne blot bliver skrevet direkte inde på chatroomet med den pågældendes navn på hvem der har afsendt det, begynder det at læne sig lidt over i et publisher-subscriber pattern, da alle der er subscribet til det chatroom vil modtage den besked som afsenderen har afsendt.  
Det er dog stadig et mediator-pattern, da man har mediatoren i form af chatroomet.

Hvis man ville gøre det mere klart at det var et mediator pattern der var med at gøre, kunne man implementere muligheden for at sende privatbeskeder til folk i chatroomet.   
Dette ville netop betyde at man kunne sende den private besked til chatroomet, som så ville afsende til den korrekte modtager.

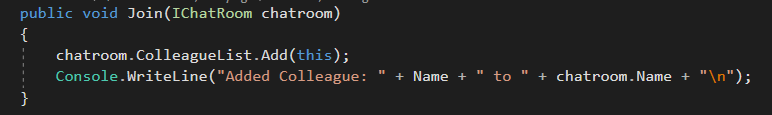
# Implementering

I opgaven er mediator pattern blevet implementeret.

Systemet er opbygget således at der er en mediator, som er et chatroom.   
Der kan godt være flere mediators, altså flere chatrooms. Der er derudover også forskellige klasser af colleague, som er oprettet for at kunne lave ændringer på den enkelte type af colleague, hvis dette er nødvendigt.

Disse colleagues kan tilslutte sig til et ønsket chatroom ved hjælp af deres funktion Join(), der tilføjer colleague til en liste af colleagues for det pågældende chatroom.

Funktionen Join() ses nedenfor.

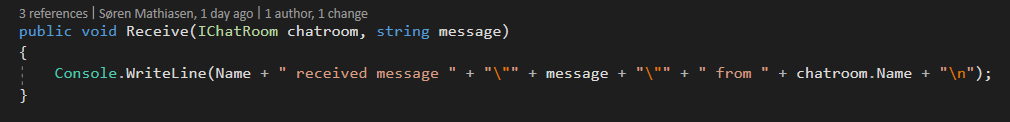


Listen af colleagues er lagt inde i IChatRoom. Se nedenfor



Når brugeren ønsker at sende en besked, kaldes funktionen Send(), der ligger i Colleague. Denne funktion kalder SendMessage(). Begge disse funktioner bliver beskrevet i næste afsnit.

Derudover har Colleague også en funktion Receive(), der udskriver hvem der har modtaget hvilken besked fra hvilket chatroom. Denne funktion ses nedenfor



# Konklusion

Mediator pattern er god til at løse problemer med høj kobling i software. Man vil gerne undgå at klasser refererer direkte til hinanden på kryds og tværs, det kan mediator hjælpe med at løse. Mediator gør det også muligt at ændre interaktionen mellem et sæt af objekter uafhængigt af hinanden. Her ændrer man blot i mediator i stedet for at ændre i de kommunikerende klasser.