# Disposition05 – MDS

## Messaging distribution system – Why and how?

Før MDS

* System med højkobling alle skal kende hinanden hvis de skal kommunikere
* Fordi at message queuen skal være kendt for at sende en besked til den
* Vi skal også vide hvordan data ønskes leveret
* Kan give problemer med cykliske includes – Hvem kommer først?
* Det er tæt forhold som ikke er nødvendigt

Efter MDS

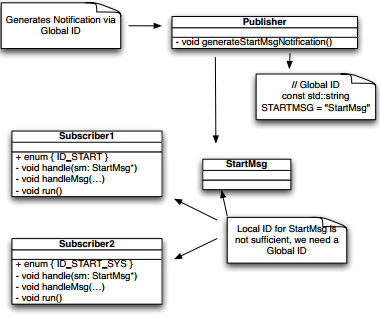
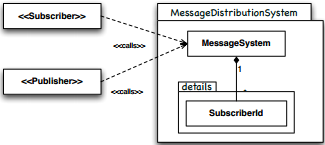
* Vi har subscribers som gerne vil have en notification når der er sket et bestemt event
* Subscribers fortæller også hvordan de vil have svaret IE hvad for et event de ønsker at få sendt
* Vi har publishers som sender notificationer igennem MDS
* MDS vider sender publishernes besked alt efter hvem der er i subscriber listen

Hvad for vi ud af det?

* Publisher skal ikke kende subscriber
* Flere subscribers kan modtage den samme besked

Krav

* MDS skal være klar til start inden brug
* Der skal benyttes en singleton eller der skal kastes pointere rundt
* Beskeder skal være globalt unikke (hos os er det gjort med strings)
* Det er kun envejs kommunikation



## The PostOffice design – Why and how?

Samme før som ved MDS post office ligner I høj grad MDS men er ikke envejs.

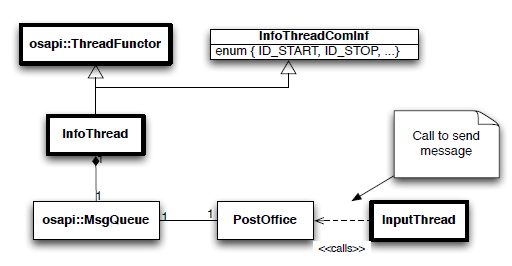
* Her er der tale om tovejs kommunikation
* Der sendes en besked til en bestemt modtager ved at navngive denne

Hvad for vi ud af det?

* Stadig lav kobling da sender modtager ikke behøver at kende hinanden (kun ved navnet i post office)
* To vejs kommunikation er muligt

Krav

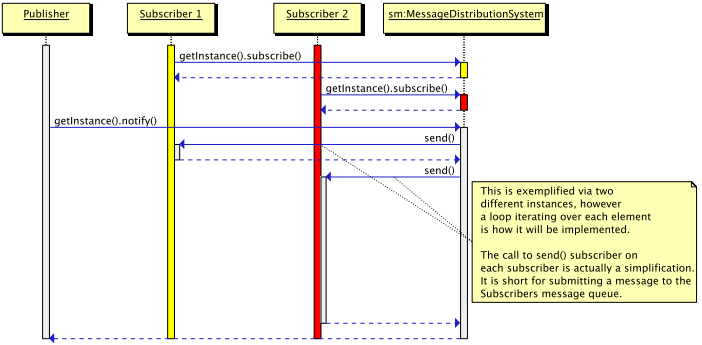
* Post Office skal køre inden brug samme argument som ved MDS.
* Singleton/pointer kast.



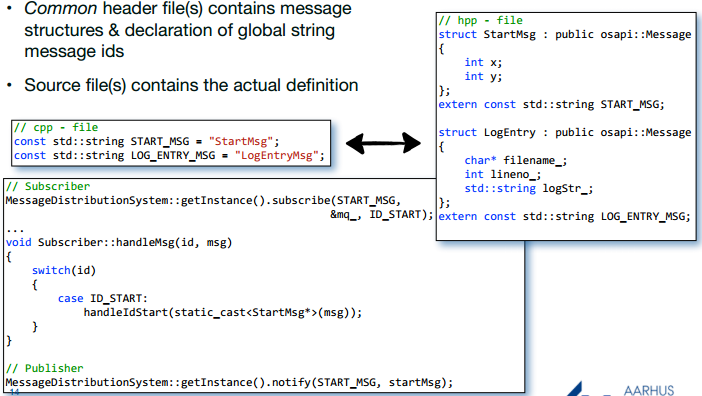
## Decoupling achieved

Afkoblingen bliver opnået ved at indsættet MDS eller post office som mellem led i kommunikationen mellem vores andre klasser.

## Design considerations and implementation



Her kan det ses hvordan MDS vil facilitere kommunikationen mellem Publishers og Subscribers.



## Patterns per design and in relation to the MDS and PostOffice design

### GoF Singleton Pattern

Udfordring

* Hele systemet skal have adgang til et objekt der er massere af pointer der skal flyttes rundt!

Mulig løsning:

* En singleton

Løser problemet ved at gøre således at der kun kan oprettes en instans af en klasse som alle så kan tilgå hos os i c++ gøres det med en statisk funktion (getInstance()) og en statisk variabel af selve klassen inde i funktionen som der returneres en reference til.

Øv:

* Den opfører sig som en global variabel den kan kun tilgåes en af gangen

Vi skal tænke på hvornår den skal oprettes og nedlægges

### GoF Observer Pattern

Udfordring

* Vi vil have besked når der sker ændringer vi vil ikke polle
* Vi har brug for at kunne sende til mange

Vi bruger Observer(Publisher/subscriber) pattern

### GoF Mediator Pattern

Udfordring:

* Vi vil gerne have et system med lav kobling hvor objekterne ikke har behov for at kende hinanden

Kunne løses med en Mediator!

* Centraliseret styring.
* Fokus er på interaktionen og ikke på opførelsen.
* Enheder behøver ikke kende til hinanden (MDS er mediator).