

Individuel Opgave

02342, DISTRIBUERED SYSTEMER

S133970 KRISTIN HANSEN

Indholdsfortegnelse

Indledning	2
Krav	3
Løsningsforslag.....	4
Server	4
Klient	4
Løsning	6
Klient	6
Server	7
Konklusion.....	9
Videreudvikling (perspektivering).....	9

Indledning

Rapporten indeholder dokumentation for den individuelle programmeringsopgave, stillet i kursus 02342, Distribuerede Systemer. Formålet med rapporten er, at redegøre for et system, som er udviklet efter publish-subscribe paradigmet.

Der skal som udgangspunkt udvikles et system, der tillader intelligent kommunikation mellem flere enheder i en bygning – styring af temperatur, underholdningsenheder, brandalarmer mv.

Mens opgaven lægger op til, at der kan udvikles et system med flere forskellige enheder, er der fortsat kun blevet udviklet grænseflader til temperaturmålere, som i de tidligere afleveringer, hvor hvert klient har en selvstændig sensor, der indsamler data og publicerer denne.

Forskellen på de tidligere systemer, og det nuværende er, at der ikke længere er gjort brug af Javas Socket-protokol. I stedet er den tidligere socket-kommunikation erstattet af RMI.

Krav

Der skal udvikles et system, der tillader en centraliseret/distribueret kommunikation via et netværk af enheder. I opgaven beskrives en intelligent bygning, som skal kunne have mange forskellige enheder, med forskellige sensorer – f.eks. brandalarmer, underholdningsenheder mv.

De indledende krav stillet i opgavebeskrivelsen kræver, at der er udvikles et middlewarelag, hvor de mange enheder kan kommunikere gennem. Herunder kræver det, at der som minimum er implementeret to centrale metoder,

- Publish
- Subscribe.

Ligeledes skal der udvikles på systemet, sådan at det er i stand til at forvalte systemets behov. Som beskrevet tidligere, bliver der stadig kun udviklet grænseflade til temperatursensorer, hvorfor den tidligere funktion "hent gennemsnit" er overført fra de tidligere afleverede opgaver.

Der er sat som krav, at der ikke længere må være nogen form for kommunikation mellem klient og server, som foregår vha. socket-kommunikation. Derfor er den tidligere socket-kommunikation erstattet af RMI.

Der er altså brug for at udvikle en server, der centraliserer kommunikationen af sensorerne, der måtte være på netværket. I denne opgave er der dog kun udviklet grænseflade til temperatursensorer. Serveren skal kunne håndtere og forvalte de abonnerede (subscribed) enheder sådan, at de er i stand til at publicere deres data på netværket. Tilgængeligheden af denne data, ville i og for sig kunne gøres tilgængelige for alle enheder, men for denne opgave, vil dataene blive gemt på serveren, for at gøre det muligt for respektive sensorer igen at hente dem (f.eks. til at udskrive et gennemsnit).



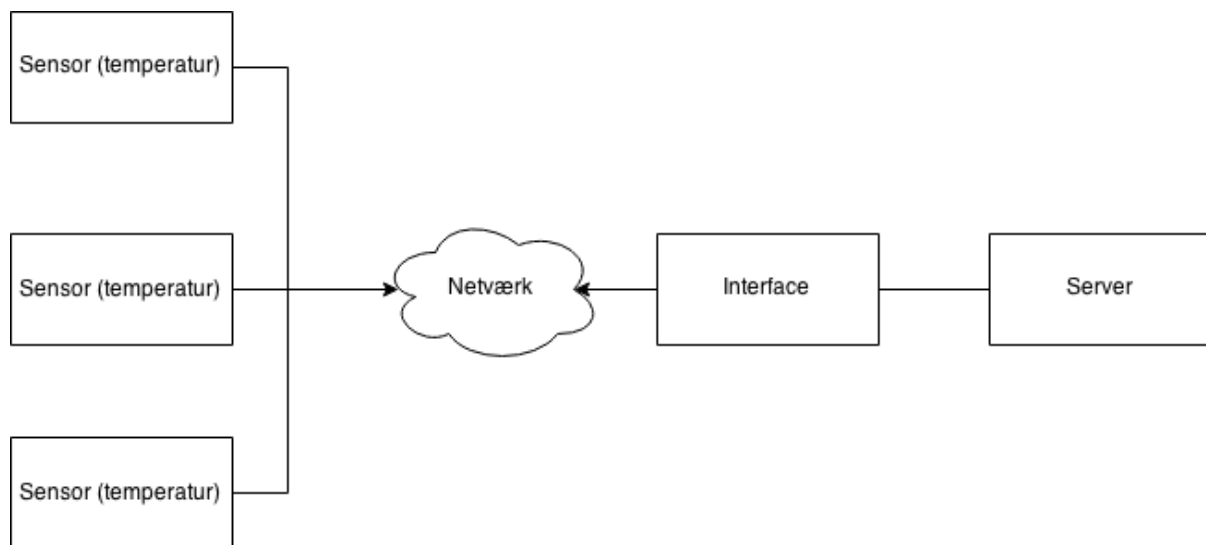
Figur 1 – Simplificeret publish-subscribe system

Løsningsforslag

Som udgangspunkt er der i opgavebeskrivelsen gjort opmærksom på, at mange forskellige typer enheder skal kunne abonnere på netværket – så der skal udvikles en subscribe-metode, der gør det muligt for forskellige typer af enheder at abonnere. På Figur 2 er der en sketchup af et sådan system, der som udgangspunkt skal udvikles, mens Figur 2 viser et mere konkret eksempel.

Server

Der skal altså udvikles en server, der kan forvalte de abonnerende enheder (sensorer), og kommunikationen til denne (serveren) skal foregå gennem et interface (i øvrigt påkrævet af RMI-protokollen). Serveren skal altså kunne håndtere flere abonnenter samtidigt.



Figur 2 – Løsningsforslag til middleware-lag

Interface er tiltænkt som kommunikationskontrakten mellem serveren og de abonnerede sensorer, hvor sensorerne implementerer denne, og tvinges til at implementere de nødvendige metoder – dette gør, at serveren er i stand til at generalisere kommunikationen, og derved centralisere denne.

Metoderne i interfacet, herunder subscribe og publish, skal være implementeret sådan, at et hvert system, af hvilken som helst type, skal kunne kommunikere gennem denne. For denne opgave er der dog sat som krav, at der ikke udvikles andet end temperatursensorer. Det ville dog være muligt at udvikle et generaliseret interface, samt specialiserede interfaces, for hver type sensor – klienterne ville kunne implementere de nødvendige interfaces, og igen gøre kommunikationen centraliseret, forudbestemt af serveren.

Serveren skal, når der abonneres, være i stand til at huske denne abonnent. Da der kun implementeres grænseflade til temperatursensorer, er det simpelt nok at oprette en liste, hvor de abonnerede sensorer løbende tilføjes. Abonnenterne skal kunne genkendes, hvilket for opgaven kunne opnås ved, at have et unikt navn for hver sensor. Havde der været behov for at implementere flere typer, kunne en generaliseret type på interfacet været implementeret, som ligeledes kunne specialiseres, og serveren ville kunne genkende disse, og tilføje (ved abonnement) sensorerne i separate lister, for hver type.

Klient

Klienten skal implementere det nødvendige interface, beskrevet på serveren, oprette de nødvendige metoder, for så at være i stand til at subscribe og publish sine data. Klienten, for denne opgave, skal

kun udvikles som en temperatursensor (der i øvrigt ikke har ændret sig siden de tidligere afleverede opgaver).

De nødvendige trin for klienten er

- Instantiere en sensor med en givet/standard temperatur.
- Subscribe til serveren.
- Udregner en tilfældig temperatur, der ligger inden for en givet/standard faktor for den nuværende temperatur.
- Publish sine data temperaturer til serveren.

Yderligere skal klienten være i stand til at hente en gennemsnitstemperatur og udskrive denne, samt være i stand til at afslutte sit abonnement.

Løsning

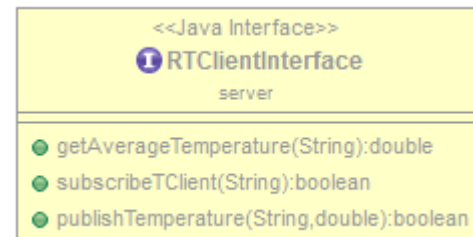
Der er fra de tidligere afleverede opgaver blevet videreudviklet et system, der nu kun gør brug af RMI-protokollen. Dvs. alle socket-forbindelser er fjernet, og i stedet erstattet af et centraliseret interface, beskrevet på serveren.

Jf. tidligere beskrivelse, er de nødvendige funktioner for systemet, at klienten skal være i stand til at subscribe, samt publish sine data. Ligeledes skal den kunne anmode om et gennemsnitstemperatur, over de sendte data. Serveren skal kunne forvalte disse klienter (sensorer), samt gemme og huske deres sendte data.

På Figur 3 ses det endelige implementerede interface. Det har de tre nødvendige metoder, subscribe, publish, samt evnen til at hente gennemsnitstemperaturen.

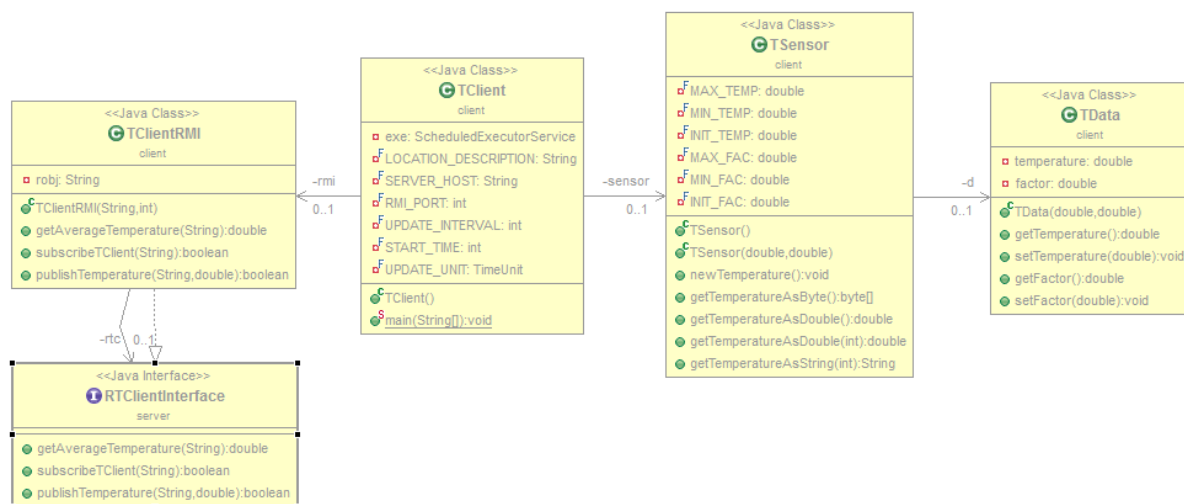
Klient

Klienten har ikke ændret sig siden seneste aflevering, men indeholder ikke længere socket-kommunikation, som er blevet erstattet af RMI.



Figur 3 – Implementeret interface

Klienten opretter en sensor, `TSensor`, som er i stand til at kalkulere temperaturen (tilfældigt genereret tal), som vist på Figur 4.



Figur 4 – Klassediagram for klienten

found..

Forskellen fra tidligere afleveringer, er at der ikke længere er nogen klasse med socket-kommunikation, og at `TClientRMI` er blevet udvidet (se Figur 4), så den nu også indeholder de nødvendige metoder til publish og subscribe. Som vist på Figur 3, returnerer de to metoder (`publishTemperature` og `subscribeTClient`) booleans, som er med til at gøre klienten opmærksom på, hvorvidt de har været succesfulde ved kald af metoderne.

Forudsætningen for, at `subscribeTClient` ikke returnerer `false`, er, at der ikke må eksistere en abonnent på serveren, der har den samme `description` (angivet med konstanten `LOCATION_DESCRIPTION` i `TClient`).

På nuværende tidspunkt returnerer `publishTemperature` `false`, hvis ikke serveren tidligere har haft registreret en abonnent, med den angivne `description`.

Klienten er stadig i stand til at kalde `getAverageTemperature`, samt gør stadig brug af en `ScheduledExecutor`, som løbende (hvert 5. sekund) kalder en ny temperatur fra `TSensor`, samt efterfølgende kalder `publishTemperature`.

Klienten har implementeret en simpel TUI, som gør brugeren i stand til at hente den seneste temperatur, samt stoppe klienten, som vist på Figur 5.

```
Sensor initialized with temperature: 21.0
Options:
1. Get average
2. Stop client
3. This message...
Choose option: |

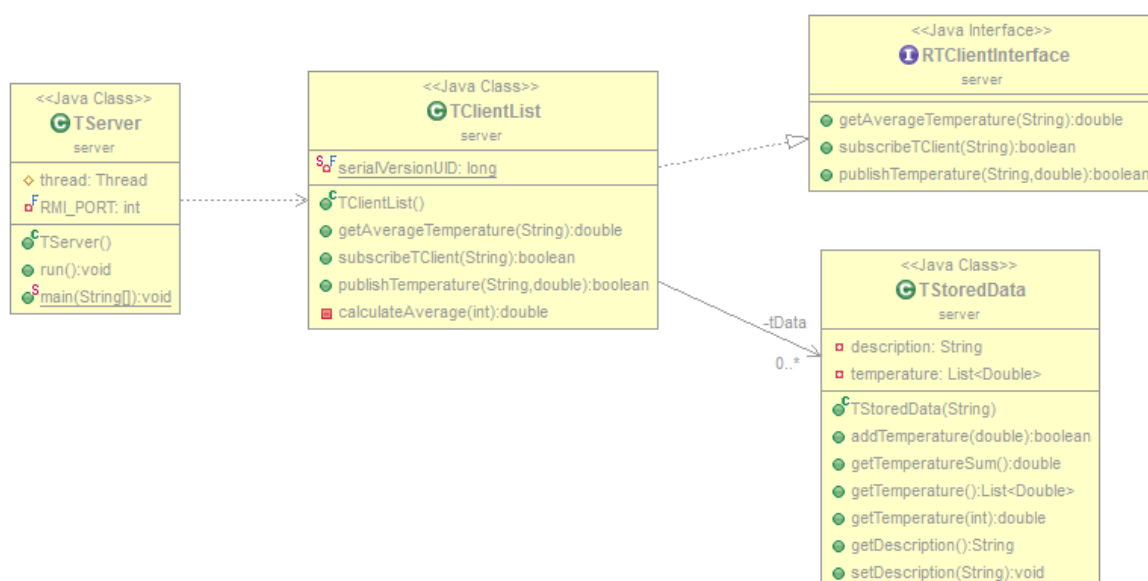
Sensor initialized with temperature: 21.0
Options:
1. Get average
2. Stop client
3. This message...
Choose option: 1
19.185882352941178
Choose option: |
```

Figur 5 – Klientens Text User Interface, samt eksempel på udprint af gennemsnitstemperatur

Server

Serveren er ligeledes siden seneste aflevering omstruktureret, så den ikke længere implementerer nogen socket-kommunikation, som i stedet er erstattet af RMI.

Interface, vist i Figur 3, er ændret så den indeholder `publish` og `subscribe`. Ligeledes er `RTClient` nu i stedet `TClientList`. `TClientList` implementerer interfacet `RTClientInterface`, og



Figur 6 – Klassediagram for server

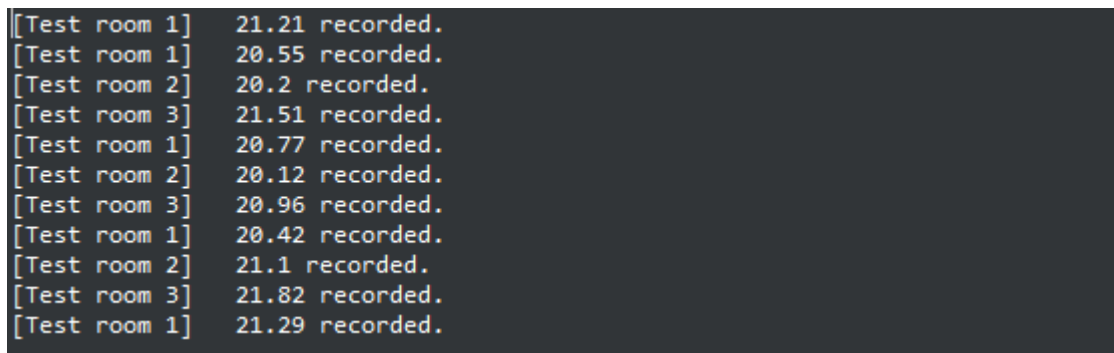
beskriver herefter metoderne, som klienten gør nytte af. Da der kun er implementeret temperatursensorer, er generaliseringen på serverens metoder lav. `TClientList` extender i øvrigt `UnicastRemoteObject`-klassen, for at gøre det muligt for klienten, at oprette en instans.

Serveren gemmer abonnenterne i en liste af typen `TStoredData` (`tData` – se Figur 6), når en klient kalder `subscribeTClient`. Eksisterer der allerede en klient i listen, der bruger samme `description`, skal der ikke tilføjes noget nyt object af typen `TStoredData`, og i stedet returneres `false`.

Metoden `publishTemperature` tillader klienten at publicere en temperatur, som vil blive gemt for den specificerede abonnent, i listen over abonnenter – der bruges navne (som beskrevet tidligere er unikke), til at genkende, for hvilken abonnent, der skal registreres en ny målt temperatur. Eksisterer der ingen abonnent med den angivne `description`, vil `publishTemperature` returnere `false`.

Serveren kan rumme så mange klienter, som hukommelsen tillader (eller indtil `List`-objektet, der gemmer klienterne bliver for stor). Ulig tidligere implementerede løsninger, er der ikke behov for særlige løsninger, hvor flere tråde skal håndteres (gør sig i øvrigt også gældende for klienten).

Serveren har i øvrigt en ikke-interagerende TUI, der blot udskriver de data, den løbende modtager fra abonnenterne.



```
[Test room 1] 21.21 recorded.  
[Test room 1] 20.55 recorded.  
[Test room 2] 20.2 recorded.  
[Test room 3] 21.51 recorded.  
[Test room 1] 20.77 recorded.  
[Test room 2] 20.12 recorded.  
[Test room 3] 20.96 recorded.  
[Test room 1] 20.42 recorded.  
[Test room 2] 21.1 recorded.  
[Test room 3] 21.82 recorded.  
[Test room 1] 21.29 recorded.
```

Figur 7 – Serverens TUI, samt eksempel på flere klienters parallelle kørsel

Konklusion

Der er blevet optimeret på et system, hvor der tidligere var blevet gjort brug socket-kommunikation, til nu kun at indeholde RMI-teknologi. Systemet fungerer efter hensigten, da alle nødvendige metoder, herunder

- Publish
- Subscribe
- Hent gennemsnitstemperatur

Er blevet implementeret, og kører som de skal.

Serveren er i øvrigt i stand til at forvalte flere klienter samtidigt. Den ønskede effekt af en centraliseret kommunikationsmodel, forvaltet af en enkelt enhed (serveren), samt muligheden for klienter at subscribe-publish på et givet netværk, er tilfredsstillende.

Videreudvikling (perspektivering)

Systemets simplificerede udgave af et subscribe-publish paradigme, bør videreudvikles sådan, at der er mere generaliserede metoder på serverens side, så den tillader flere typer af enheder (f.eks. brandalarmer, underholdningsenheder mv.)

Ligeledes er der småting i det nuværende system, som kunne udbedres – f.eks. er serveren ikke i stand til at fjerne eventuelle gemte data, skulle en abonnent afbryde/desubscribe. Der er heller ikke implementeret nogen sikkerhed – dette ville være opnåeligt, ved en simpel implementering af en SecurityManager (som er inkluderet i Javas RMI-protokol).

Om ikke andet, ville det nuværende system umiddelbart godt kunne bruges som skelet til videreudvikling af et mere sofistikeret system (intelligent, om man vil).