02342 Distribuerede systemer Assignment 3

“Individuel programmerings opgave”

Rúni Egholm Vørmadal s134004

Indledning

Jeg har anvendt koden fra de tidligere opgaver som grundlag for denne opgave, dog med mindre ændringer. De involverede i de tidligere opgaver var: Christian Budtz s134000 og Casper Thøro Vium Pedersen s133961.
Jeg har videreudviklet serveren sådan at den understøtter publisher/subscriber paradigmet og ændret protokollen, så den passer bedre til den nye implementering. For overskuelighed har jeg også lavet en GUI til både server og sensor, som indeholder en log, så man nemmere kan følge med hvad der sker. RMI implementeringen fra sidste opgave er uændret.

Kravspecifikation

Systemet skulle ændres sådan at kommunikation mellem sensorer og server skulle foregå ved hjælp af publish/subscribe paradigmet, hvilket giver løst koblet interaktion mellem system delene. Systemet skal understøtte publish(event) og subscribe(event), hvor en logisk tilføjelse er unsubscribe(event).
Kommunikationen mellem brugerens grænseflade og server skal stadig håndteres med RMI. Der blev ikke stillet nogen FURPS (Functionality, Usability, Reliability, Performance, Supportability) krav.

Analyse

I et intelligent hus kan Publisher/Subscriber paradigmet implementeres på mange forskellige måder. Huset har højst sandsynligt mange forskellige sensorer, som laver målinger i forskellige intervaller og har forskellige krav specielt mht. ‘reliability’, men også ‘functionality’ og ‘performance’.

Reliability

Systemet skal kunne håndtere nedbrud af sensorer og servere, og hvis systemet skal rumme sensorer som fx alarmer: røgalarmer, tyverialarmer, eller sensorer som kan fx. åbne garage døre, åbne vinduer e.l. så skal systemet kunne garantere dataoverførslen. Altså have for eksempel en TCP forbindelse.
Man vil gerne sikre at subscribe events og unsubscribe events når frem, men også at subscribers unsubscriber i tilfælde af nedbrud. Ved at bruge TCP forbindelse til disse events løses første problem, og ved at lave en slags check-in med jævne mellemrum, er subscriptions altid up-to-date. Dette giver noget overhead, men sørger for at data ikke sendes uden at have en destination. For det meste af tiden. Man kan også nummerere eventsne og hvis en event ikke når frem, kan man bede om at få den sendt igen. Her kan man også få rækkefølgen rigtig, hvis det skulle være af stor betydning.

Functionality

For at øge om fleksibiliteten kan man implementere en løsning som ikke er teknologiafhængig. Man vil gerne have forskellige slags platforme skal kunne snakke

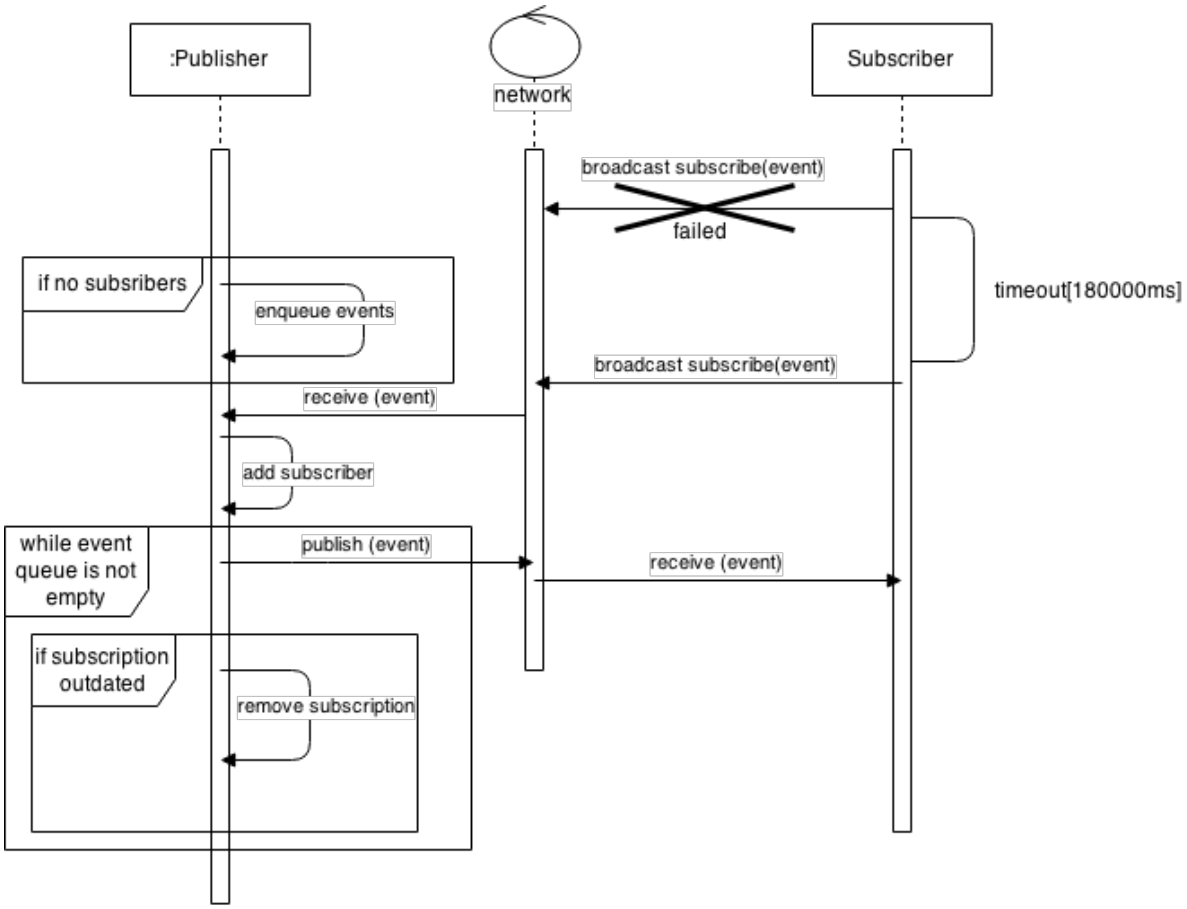
sammen. Derfor er en løsning med RMI ikke så god, da dette kræver en java virtual machine kører på enheden.

Performance

Sensorer kører ofte på batteri, og derfor er det vigtigt at begrænse datatrafikken så meget som muligt, da det er en storforbruger af strøm. En opsamlingssensor som for eksempel temperatur måler, sender ikke meget data til at dette kan blive et problem på netværket. Man kan have et selvstændigt netværk, som sensorerne kører på, for at undgå at andre forbindelser skaber stor belastning på netværket og dermed får stort pakketab.

Design

Jeg har valgt en så teknologi uafhængig løsning som muligt, da jeg vil lave et system som har stor fleksibilitet og langt levetid, fx er RMI en udgående teknologi, som også kræver at det samme er på begge sider af forbindelse. Jeg har valgt en simpel løsning hvor hver node, publisher (sensor) eller subscriber (server), melder sin ankomst ved opstart, med at broadcaste en besked. I tilfælde af en ny subscriber broadcaster den en subscription med valgte topic. Sensorerne tilføjer subscriberen på sin egen liste af subscribers og publisher derefter til alle adresser på denne liste. Hvis listen af subscribers er tom, tilføjes målinger til en kø, indtil en subscriber er fundet. Derved sikres at minimum en subscriber får publisher events. Undtagelsen er dog hvis subscriberen går ned uden at publisheren bliver gjort opmærksom på dette. Her har jeg lavet en løsning hvor subscriberen sender subscriptions med jævne mellemrum, se figur 1. Af test grunde sendes subscriptions med et interval på 10 sekunder, et mere passende interval ville være måske et par timer. Subscriptions hos publisheren har så en levetid som er 3 gange længere end de intervaller subscriptions bliver sendt fra subscriberen.
Hver gang en subscription bliver modtaget hos publisher bliver den tilføjet en liste, men hvis den allerede eksisterer bliver levetiden bare opdateret. Sådan undgås at data bliver sendt flere gange til den samme subscriber, men også undgås at gamle subscriptions hobes op hos publisheren. Jeg har også lavet en unsubscribe event for god ordens skyld, men subscribers bliver automatisk unsubscribed hvis de holder op med at sende subscriptions til publisheren.
Som ekstra funktionalitet bliver events fra publisher gemt i en liste, hvis ingen subscriptions er modtaget. Når en subscription modtages, tømmes listen, og derved kan målinger laves, før en subscriber er fundet.



figur 1. sekvensdiagram over hvordan broadcast af subscription i intervaller af 3 minutter sørger for at subscriptions når frem. Publisheren er allerede startet med at lave målinger, men da subscribe eventet ikke er nået frem, sættes eventet i en kø.

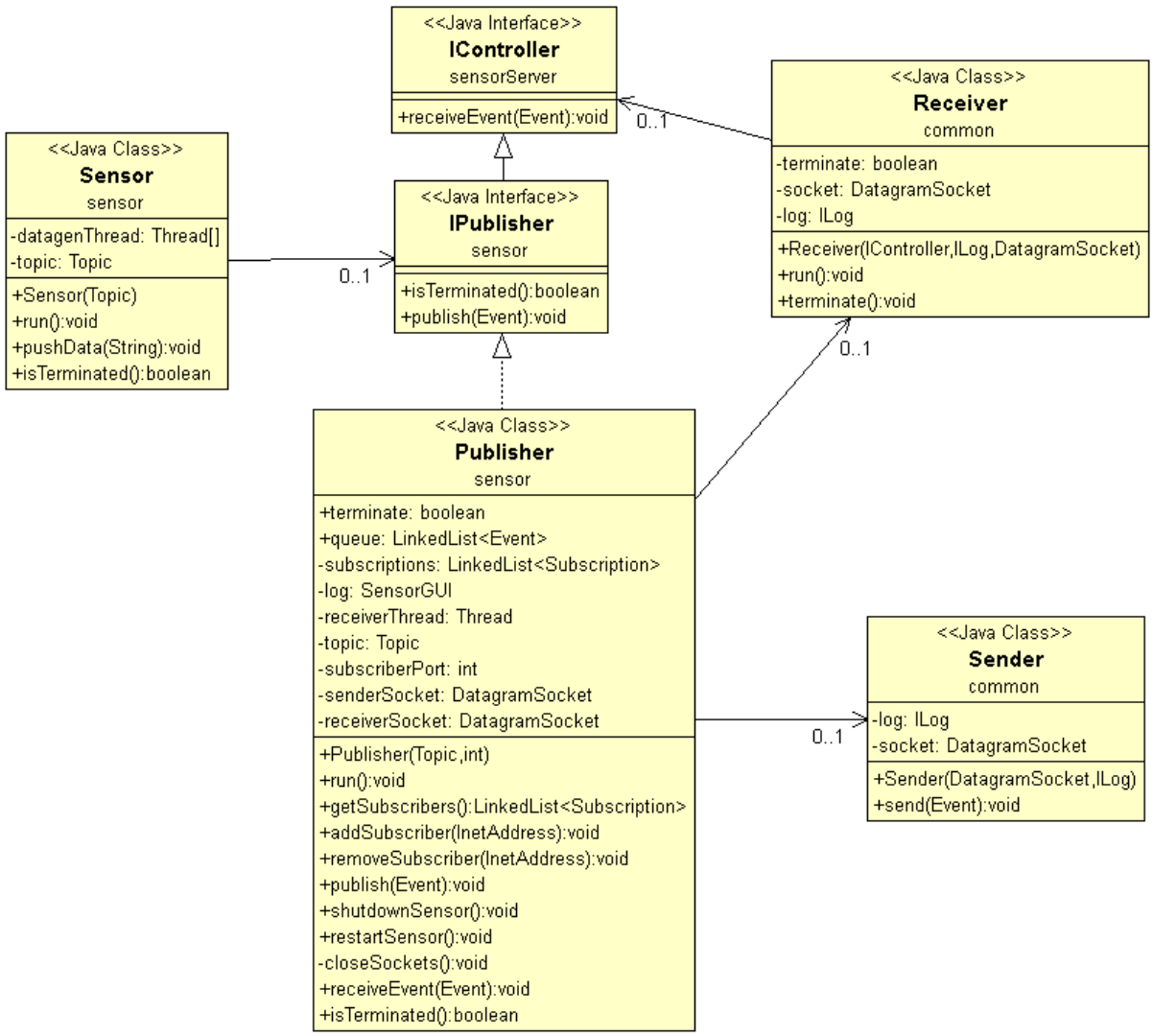
Jeg har lavet en protokol hvor man skal sætte topic, en værdi og et portnummer adskilt med semikolon (TOPIC;VALUE;PORT;). Port nummeret angiver hvilken port publisheren lytter på til subscriptions af angivne topic. Dette gør at man undgår at skulle slå op, hvilken port hvert topic tilhører. Subscriberen lytter altid på samme port, 8888, mens publishers lytter på den port angivet af sit topic. Value fortæller om det er et subscribe, unsubscribe eller publish event. Subscribe og unsubscribe event har en predefineret tekststreng for 'value', mens publish events kan være alt andet, derved bliver alle pakker som ikke er subscribe eller unsubscribe læst som publish events. Det er så op til modtageren at bestemme om 'value' har det rigtige format. Her ændrede jeg formatet på målinger fra tidligere opgaver til kun at opfylde det regulære udtryk: "\d{2}\.\d{2}". To tal (0-9) efterfulgt af et "." efterfulgt af to tal. I andre tilfælde ville man dog ikke være interesseret i hvilket format 'value' har, men da der her skal laves beregninger på værdierne er formatet vigtigt.

Implementering

Sensor (publisher)

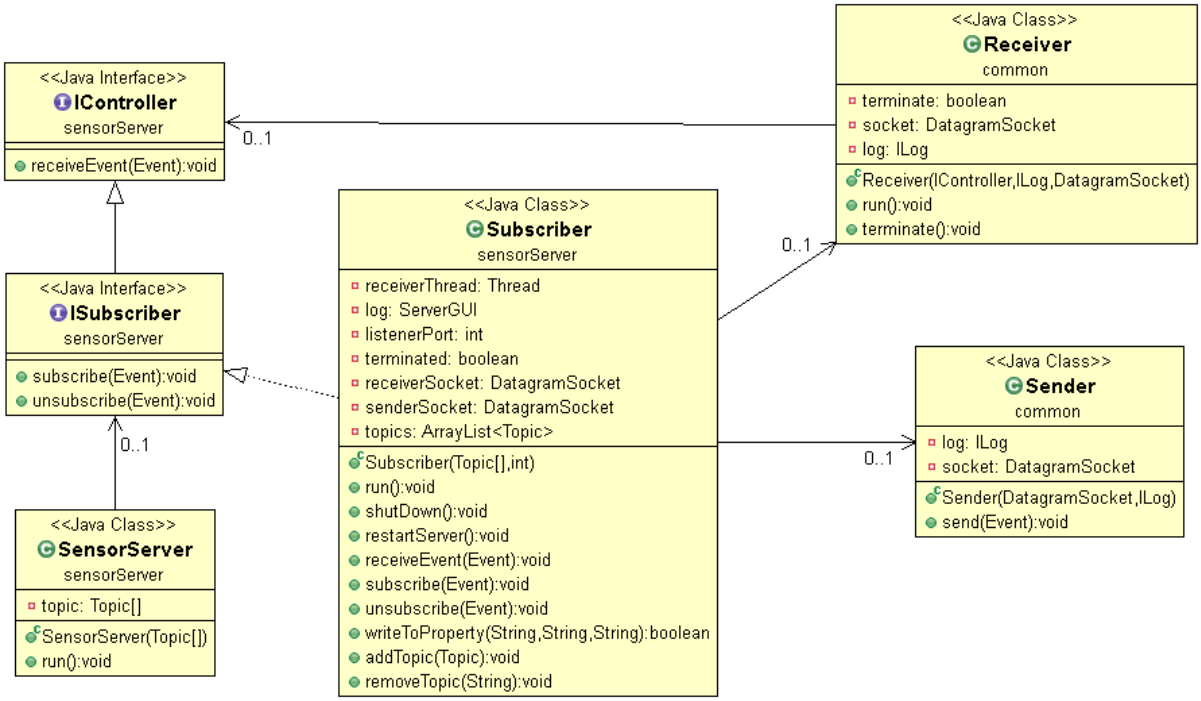
Der er udstillet et IPublisher interface som Publisher implementerer. Sensoren fungerer som en kontroller, hvor data genereres og bliver pushet til publisheren, ved publish

metoden. Dette er lavet sådan at Sensor objektet nemt kan udskiftes og dermed gør det nemmere at genbruge Publisher, da der ikke er nogen afhængigheder mellem disse. Publisher har to tråde til data, en Sender og en Receiver. Udladt af diagrammet er gui'en, da denne kun var til for nemmere testing. Publisher modtager et topic, i form af en identifier og portnummer, og et portnummer, som angiver porten subscriberen lytter på. Publisher sørger så for at at subscribers bliver tilføjet en liste, og at publishe til alle på listen. Ved opstart broadcaster publisheren også en 'READY' besked. Da alle subscribers er på samme port skal der kun broadcastes en gang. Publisher opdaterer også listen af subscribers efter hver publish og fjerner uddaterede subscriptions. Publisher sørger også for at sætte events i kø, hvis listen af subscriptions er tom. Sensor objektet kan erstattes med hvilket som helst andet objekt som skal publishe, da Publisher ikke er afhængig af nogen sensor.



Server (subscriber)

SensorServer instantierer en Subscriber som implementerer ISubscriber. Som ligesom Publisher har en Receiver og Sender i hver sin tråd. Data som modtages bliver pt. gemt af Subscriber. Subscriber kan have flere topics, og kan tilføje og fjerne topics som det passer. Subscriber checker format af data som modtages, og gemmer i en fil, navngivet af topic.



Test

Jeg har testet på en lokal maskine at server og sensor kan startes i tilfældig rækkefølge. Sensorerne kan ikke starte, hvis det port de er tiltænkt allerede er i brug, men dette stopper ikke hele systemet men sensoren, som kan restartes efter at porten er lukket, og så kører alt igen.

Konklusion

Jeg har bygget et simpelt Publisher/subscriber system, som kan håndtere sensorer som ikke stiller krav til at data når frem. Med temperatur sensorer som udgangs punkt. Jeg har opfyldt kravet om at have en publish(Event) og subscribe(Event) metode, samt at jeg har tilføjet en unsubscribe(event) metode.

Brugervejledning

For at køre programmet skal man køre følgende mains som ligger i default package:

1. ServerMain.java starter både RMI server og server til sensorer. En brugergrænseflade popper op, hvor information om subscriber (serveren) logges.
2. SensorMain.java starter 10 sensorer som hver sender tilfældige værdier mellem 14 og 24 med et mellemrum på 3 sekunder, indtil de lukkes.
3. ClientMain.java starter en grafisk brugergrænseflade, hvor man kan hente gennemsnit af alle målinger. Hvis serveren ikke er startet kan man få den til at forsøge at reconnecte.

Værdier gemmes i en properties fil kaldt temperature.properties som oprettes i projektfolderen, som kan slettes/tømmes hvis man vil gøre et nyt forsøg.