Corso di Programmazione Concorrente e Distribuita

Relazione Assignment #3

Simulazione di un sistema di monitoraggio dei possibili allagamenti in una smart-city

Alessandro Magnani, Simone Montanari, Andrea Matteucci

Analisi del problema

L’obiettivo del programma è la realizzazione di un’applicazione distribuita per monitorare possibili allagamenti in una smart-city, posizionando sensori in varie zone della città che potessero avvertire le rispettive caserme dei pompieri in caso il numero di allarmi all’interno dell’area supervisionata superassero una determinata soglia.

Il gruppo ha deciso di utilizzare un MOM (l’implementazione che fornisce RabbitMQ) sfruttando il broker di messaggi per simulare valori randomici dei livelli di pioggia, per gestire gli allarmi e le varie reazioni dei componenti.

È di seguito riportato uno screenshot del sistema in esecuzione.

Come è possibile osservare dall’immagine, la smart-city è stata suddivisa in aree ognuna supervisionata da una rispettiva caserma dei pompieri. Quando il numero di allarmi notificati dai sensori (in rosso) e generati da un valore pseudorandomico, supera la metà dei sensori totali, viene notificata la caserma del problema occorso che, attraverso l’interfaccia relativa al pannello di controllo, potrà risolvere la situazione, riportando i sensori allo stato originario.

Architettura proposta e strategia risolutiva

All'interno del progetto sono presenti i seguenti file:

- *SimulationView*, il componente grafico dell'applicazione
- *Controller*, il controller che svolge la funzione di intermediario tra View e Subscriber
- *Subscriber*, classe utilizzata per dichiarare i receiver e le loro callback
- *Publisher*, classe utilizzata per pubblicare valori randomici dei livelli di pioggia

Il gruppo ha deciso di implementare un exchange di tipo *direct*, sfruttando la possibilità di ricevere messaggi in modo selettivo specificando un'opportuna routing key.

Sono state realizzate due tipi di code separate (*queueNameSensor* e *queueNameFirestation*) relative ai due tipi di Subscriber presenti nell'applicazione:

- Sensori
  Siccome in ogni area sono presenti vari sensori e ogni singolo sensore deve ricevere valori randomici del livello di acqua, si è deciso di istanziare un subscriber per ogni sensore (la routing key utilizzata è X-Y, dove X è l'area di appartenenza del sensore, e Y il numero del sensore di modo da identificare ed associare ogni messaggio al sensore corretto).
  Di conseguenza, ogni sensore avrà una propria *queue* e una propria *routing key.*

- Caserme dei pompieri
  In ogni area della smart-city è presente una caserma che riceverà segnali di allarme ogni volta che la maggioranza dei sensori nella sua rispettiva area sono in stato di allarme (infatti ogni caserma gestirà esclusivamente una sola area, individuabile in quanto contrassegnate dallo stesso numero).
  A tal proposito, similmente ai sensori, ad ogni caserma sono assegnate una queue e una specifica routing key rappresentata da un semplice numero indicante solamente la sua specifica area di riferimento.

Sono state implementate due diverse callback (*sensorCallback* e *firestationCallback*) per gestire i comportamenti dei sensori alla ricezione dei livelli di pioggia e, conseguentemente, delle caserme alla ricezione di possibili allarmi.
Nello specifico ogni sensore, alla ricezione di un valore sopra a una soglia pre-determinata, si mette in stato di allarme (colorandosi di rosso).
Inoltre esegue un controllo sugli altri sensori all'interno della sua zona e, se la

maggioranza sono in allarme, invierà un allarme alla caserma dei pompieri che ha in gestione quella determinata area.

Alla ricezione di un messaggio di allarme, la sezione relativa alla caserma nel pannello di controllo in basso si colorerà di rosso e verrà attivato il pulsante 'Manage Alarm', in attesa che un operatore 'inizi la manutenzione' in quella determinata zona.

L'area in questione entrerà temporaneamente in uno stato di busy (indicato sia nel titolo della GUI relativa alla zona, che nel pannello di controllo nella parte specifica dell'area e in quella riassuntiva in basso a destra), mentre i sensori che avevano rilevato l'innalzamento di acqua si coloreranno di giallo, come ad indicare il processo di manutenzione che sta avvenendo loro, uno stato intermedio tra la segnalazione di errore ed il risolvimento del problema.

Una volta terminata questa fase, la caserma relativa all'area manutenuta, sempre attraverso il pannello di controllo, potrà premere il pulsante 'Resolve Alarm' che permetterà di far tornare sensori ed area alla situazione originaria (ovvero di modo che indichino uno stato dell'acqua consono agli standard naturali).
In pratica, a livello grafico l'area nel pannello di controllo tornerà free e di colore verde così come i sensori torneranno di colore grigio.

Comportamento del sistema

È di seguito riportato il comportamento del sistema:

1. Inizialmente occorre avviare il main presente all'interno della classe ***Subscriber***.
   Una volta avviato, vengono svolte le seguenti operazioni:
   - viene istanziato il componente grafico (*SimulationView*)
   - vengono generati i parametri per gestire il MOM come il channel di cui verrà esplicato l'exchange (di tipo direct)
   - viene generata la callback per i sensori
   - viene generata la callback per le caserme
   - viene generato il legame tra exchange e code di sensori e caserme specificando le relative routing key (*queueBind()* e *basicConsume()*)

   A questo punto la componente grafica e le varie code dei subscriber sono in attesa di messaggi dal *Publisher*.

2. Successivamente, occorre avviare il main presente all'interno della classe ***Publisher*** per poter generare valori randomici di livelli di pioggia e mandarli ai vari sensori nella città.

3. Il ***Subscriber*** (attraverso la callback dei sensori e la callback delle caserme, rispettivamente *sensorCallback* e *firestationCallback*) provvederà a cambiare di color rosso sia i sensori il cui valore randomico è superiore ad una certa soglia preimpostata che l'intera area all'interno del pannello di controllo in cui la maggioranza dei sensori è di color rosso (per determinare la maggioranza è stato generato il metodo *checkForOtherSensor()*).

4. Una volta che l'area è in stato di allarme, viene sbloccato il pulsante 'Manage Alarm' relativo alla caserma che gestisce l'area in questione, che sblocca il pulsante 'Resolve Alarm' e cambia lo stato dell'area in Busy e il colore dei sensori da rosso a giallo (attraverso l'*actionListener* del *manageButton* della *SimulationView*).
   Come da specifiche, un sensore in allarme il cui nuovo valore randomico al ciclo successivo è inferiore alla soglia, non muta il colore in grigio, bensì rimane colorato di rosso.

5. Una volta che l'area è occupata per la manutenzione, se si preme il pulsante 'Resolve Alarm', l'area torna Free e di colore verde mentre il colore dei sensori torna grigio (attraverso l'*actionListener* del *resolveButton* della *SimulationView*).

Conclusioni

La realizzazione e, soprattutto, il processo di brainstorming per elaborare le due strutture relative alle due parti, sono stati molto interessanti in quanto il gruppo ha dovuto cambiare metodologia di pensiero.
A differenza degli scorsi assignment dove i ragionamenti erano Thread e Task oriented, in questo si è dovuto ragionare con attori che si scambiano messaggi.
Abbiamo incontrato le maggiori difficoltà nel capire come riuscire a gestire il protocollo dei messaggi per far sì che gli attori reagissero in un determinato modo ad un determinato messaggio.

In conclusione, riteniamo che cambiare metodologia di ragionamento sia stato utile per comprendere più a fondo come occorre gestire sistemi basati sullo scambio di messaggi.