Relazione PCD assignment 3

Denys Grushchack

Dicembre 2022

Indice

Capitolo 1

Primo esercizio

Per semplicità omettiamo le fasi di analisi, dato che sarebbero congruenti a quelle fatte per il primo assignment, fatta eccezione delle conclusioni che verranno riformulate nel paragrafo successivo.

1.1 Strategia risolutiva e architettura proposta

Il sistema prevede tre tipi degli attori

- `CoordinatorActor` crea e imposta altri tipi degli attori. Ha un behavior iniziale e due behavior ciclici:
  - `UpdateVelocityCoordinatorBehavior` raccoglie in un buffer risultati parziali dell'aggiornamento delle velocità. Quando tutti i risultati sono raccolti prima viene richiesto ai worker di aggiornare le posizioni degli corpi e dopo il behaviour del attore viene cambiato.
  - `UpdatePositionCoordinatorBehavior` raccoglie in un buffer risultati parziali dell'aggiornamento delle posizioni. Quando tutti i risultati sono raccolti le coordinate dei body vengono inviate a `ViewActor` per la visualizzazione (se `ViewActor` esiste). Se è stata ricevuta una conferma che l'iterazione precedente della simulazione è già visualizzata dalla gui, allora il `CoordinatorActor` inizia una nuova iterazione e cambia il suo behavior, altrimenti aspettiamo la conferma per poter procedere.

- `WorkerActor` ha il compito di aggiornare i stati dei corpi della simulazione. Come il suo padre ha un behaviour iniziale e due behaviour ciclici:
  - `VelocityUpdateWorkerBehaviour` per una partizione dei corpi: calcola la forza, l'accelerazione e conseguentemente aggiorna la velocità del corpo.
  - `PositionUpdateWorkerBehaviour` per una partizione dei corpi: aggiorna la posizione dei corpi e controlla i bordi del campo.
- `ViewActor` esiste solo se il programma viene eseguito con una GUI. Visualizza i corpi e invia un ack per confermare la fine della visualizzazione.

Per garantire il rispetto delle dipendenze temporali, prima di attivare il comportamento `UpdatePositionCoordinatorBehavior` del coordinatore, tutti i `WorkerActor` devono completare il calcolo ed aggiornamento delle velocità e mandare i risultati parziali al `CoordinatorActor`.

lo Start, come negli assignment precedenti, è stato realizzato riavviando l'intera simulazione, mentre lo Stop funziona, in caso di pressione del tasto, terminando l'`ActorSystem` che permette ricorsivamente terminare tutti gli actor in esecuzione.

1.2 Descrizione del comportamento del sistema

Qui di seguito viene riportata la rappresentazione di un'esecuzione della simulazione mediante gli attori.

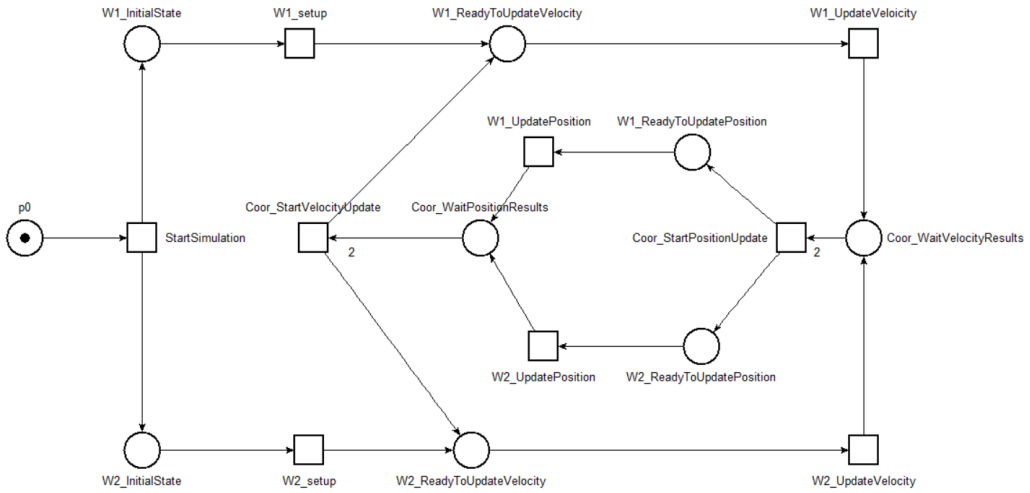

Figura 1.1: Rete di Petri semplificata che rappresenta il comportamento del sistema che prende in considerazione due `WorkerActor` (`W1_` e `W2_`) e un `CoordinatorActor` (`Coor_`)

Nella fig. 1.1 possiamo vedere una rete di Petri che mostra il funzionamento della simulazione basata sugli attori. Avendo utilizzato gli stessi principi di suddivisione del lavoro utilizzati nei primi due assignment la rete di Petri risulta molto simile a quelle proposte in precedenza.

Nella fig. 1.2 possiamo vedere un fragmento della rete di Petri mostrata precedentemente (fig. 1.1), questo fragmento inquadra la situazione che precede l'inizio di una nuova iterazione della simulazione. Si può notare che il coordinatore non può far partire il nuovo ciclo affinché non riceve tutti i corpi con le posizioni aggiornate e una conferma dalla view che i corpi nella iterazione precedente sono già visualizzate.

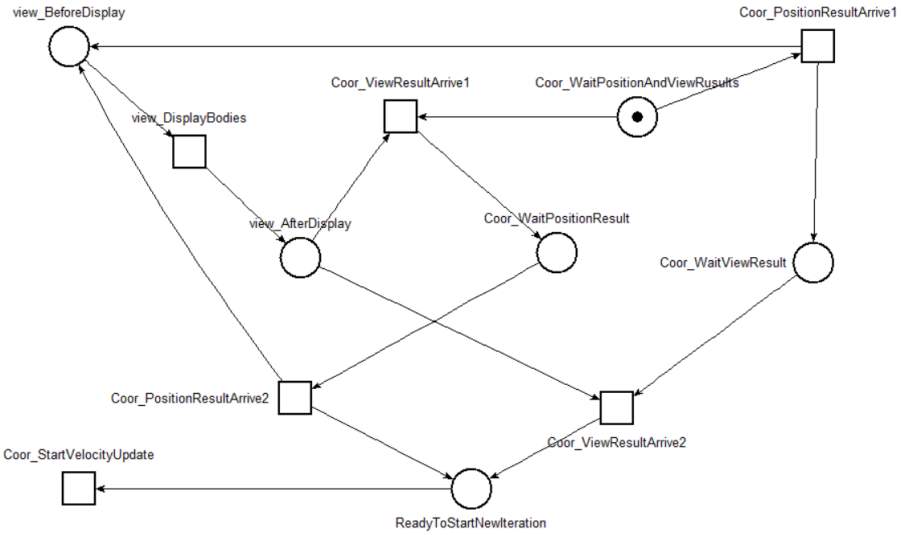

Figura 1.2: Rete di Petri che modella in maniera più dettagliata le iterazioni tra `CoordinatorActor` e `ViewActor` (`view_`) prima e dopo l'arrivo dei corpi con le posizioni aggiornati

1.3 Prove di performance

Le comparazione di tempi d'esecuzione e di speedup dei tre versioni del programma si può vedere nella fig. 1.3, le misurazioni sono fatte su un computer 4 core (8 core logici). Dai risultati possiamo notare che:

- Nel caso con 100 body abbiamo notato che lo spedup della versione ad attori è leggermente migliore rispetto alla versione a task, ma solo per il numero di thread/attori minore di 8 ma con un numero maggiore di essi lo speedup di due versioni converge. Tra altro la versione ad attori mostra i migliori risultati che riguardano i tempi d'esecuzione assoluti.
- Nel caso con 1000 body la simulazione ad attori mostra le prestazioni peggiori per entrambe le metriche.
- Nel caso con 5000 body le performance ottenute nei tre approcci sono molto simili in quanto la logica di suddivisione del lavoro è essenzialmente la stessa anche se le architetture utilizzate sono diverse.

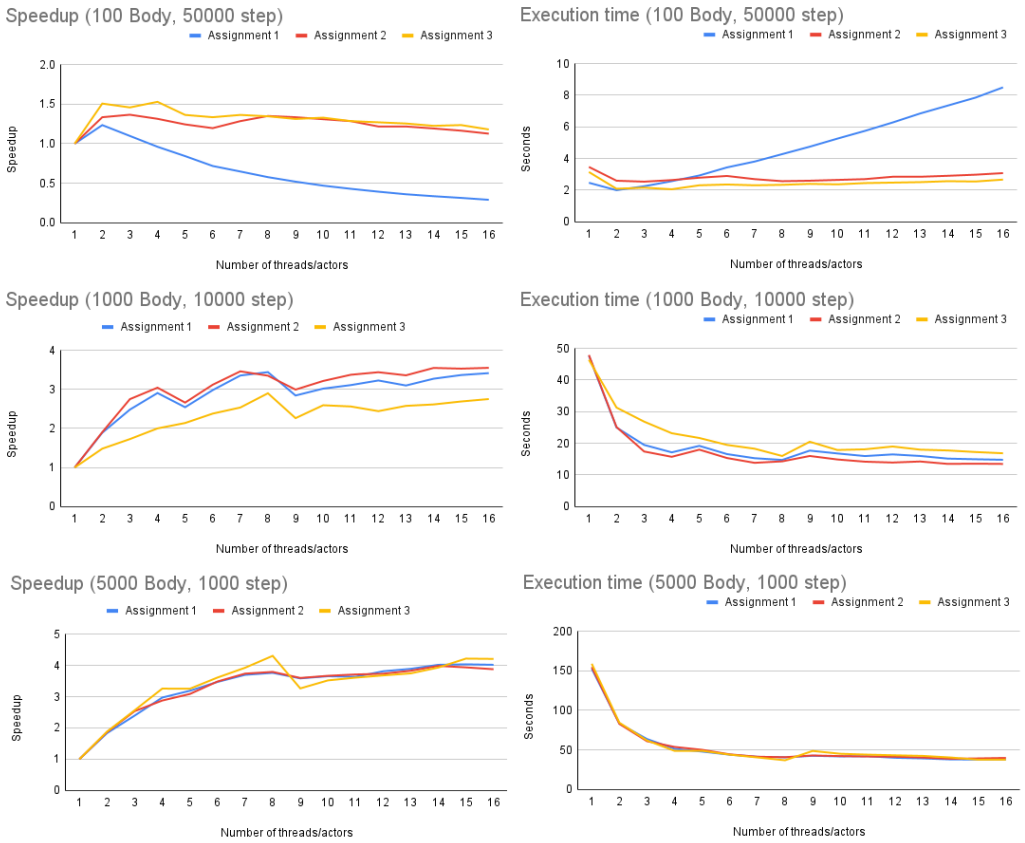

Figura 1.3: Comparazione dei tempi assoluti e degli speedup delle tre versioni della simulazione

Capitolo 2

Secondo esercizio: smart city

2.1 Analisi del problema

L'obbiettivo dell'esercizio è creare un'applicazione distribuita che ha lo scopo di monitorare possibili allagamenti in una smart-city. La città è rappresentata da una mappa divisa nelle zone. In ogni zona c'è un insieme di pluviometri (sensori) che comunicano tra loro in una rete p2p. I sensori nelle zone diverse non comunicano tra loro e sono completamente isolati. In ogni zona c'è una caserma dei pompieri con una GUI che permette visualizzare la mappa e i dati rilevati dai sensori della sua zona, in caso d'allarme nella zona l'applicativo permette disattivarlo.

Quando la maggioranza dei sensori nella zona supera una soglia del livello dell'acqua la zona va nello stato d'allarme. I sensori sono in grado autonomamente determinare se la maggioranza ha superato la soglia d'allarme. Durante esecuzione il numero dei sensori in ogni zona può variare e quelli attivi devono correttamente reagire a tali cambiamenti.

2.2 Architettura proposta

2.2.1 Caserma dei pompieri

Considerando che ogni caserma dei pompieri deve avere una interfaccia grafica per visualizzare la mappa della città noi raggruppiamo tutte le caserme (`FireStation`) in un unico gruppo, ogni `FireStation` avrà seguenti compiti:

- Fornire una interfaccia grafica
- Ricevere dati raccolti da tutti sensori della città
- Visualizzare sulla mappa le posizioni e gli stati dei sensori

- Visualizzare il livello dell'acqua rilevato dai sensori nella sua zona
- Mostrare se la zona è in allarme o no
- Permettere disattivare tutti gli allarmi dei sensori nella sua zona (diminuendo la quantità d'acqua)

Per rendere il software più flessibile di default la mappa non visualizza i sensori, solo quando un sensore viene attivato e si presenta ai `FireStation` fornendo le sue coordinate quello dipinta un nuovo sensore sulla mappa.

2.2.2 Pluviometri

Il sensore può essere in due stati principali "OK" e "allarme locale" ma indipendentemente da quelli può entrare nella stato "allarme della zona" se il leader della zona rileva che la maggioranza dei sensori sono nello stato d'allarme locale.

Elezioni

Elezioni del leader da punto di vista di un singolo sensore è mostrato nella fig. 2.1. Al inizio il sensore si candida come il leader mandando a tutti gli altri sensori il suo id. Quando a un nodo arriva id di un candidato questo nodo lo confronta con id del suo leader, se id del candidato è minore allora il leader viene cambiato, altrimenti un nodo ordinario ignora il messaggio mentre il leader effettivo risponde al candidato con la sua auto candidatura.

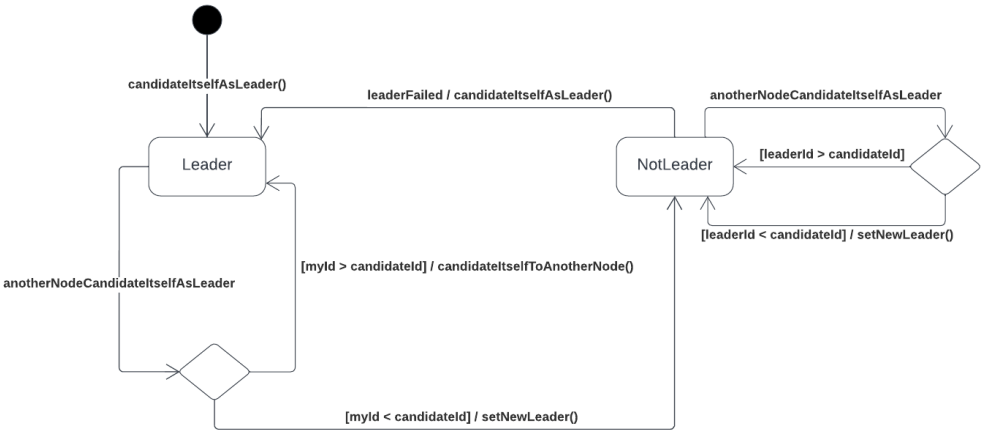

Figura 2.1: Diagramma degli stati di un sensore durante elezioni del leader

Cambiamenti degli stati di un sensore

Nella fig. 2.2 è mostrato come un sensore cambia il suo stato d'allarme locale e della zona. All'inizio un sensore non ha nessun allarme e manda con un periodo le misurazioni del livello dell'acqua ai `FireStation`. Da questo stato il sensore può andare direttamente nello stato d'allarme della zona se il leader lo ordina oppure se il sensore rileva un livello anomalo dell'acqua dovrà entrare nello stato d'allarme locale notificando il leader. I sensori che sono nello stato d'allarme della zona mandano i dati misurati alla caserma dei pompieri specificando che sono nello stato d'allarme cosi l'applicativo può visualizzare quali sensori hanno raggiunto il consenso.

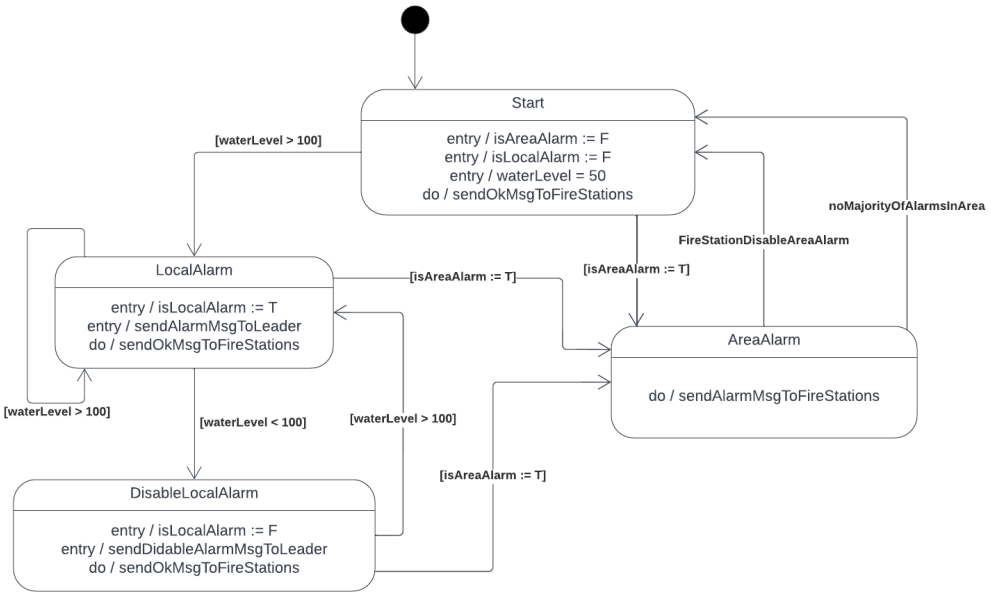

Figura 2.2: Diagramma degli stati di un sensore

Un sensore leader della zona registra il numero dei sensori che sono nello stato dell'allarme locale e se c'è una maggioranza ordina ai sensori di entrare nello stato d'allarme della zona. Se il leader viene cambiato, ogni nodo disattiva il suo allarme locale e globale cosi il nuovo leader potrà rivalutare se è necessario far entrare di nuovo nello stato d'allarme della zona.

2.3 Implementazione

Per implementazione è stato scelto utilizzare l'approccio ad attori utilizzando il framework Akka, L'applicazione è completamente realizzata nel linguaggio scala.

Le GUI di ogni caserma dei pompieri vengono eseguite nei processi separati. Anche ogni sensore richiede l'esecuzione in un processo separato. Per semplificare l'avvio della applicazione è stato scelto creare solo due zone con 3 e 1 sensori rispettivamente.

Tutti i nodi del sistema appartengono allo stesso cluster Akka, tramite receptionist è stata effettuata una suddivisione dei nodi nei gruppi. Le caserme dei pompieri condividono lo stesso gruppo mentre i sensori di ogni zona hanno il loro proprio gruppo.

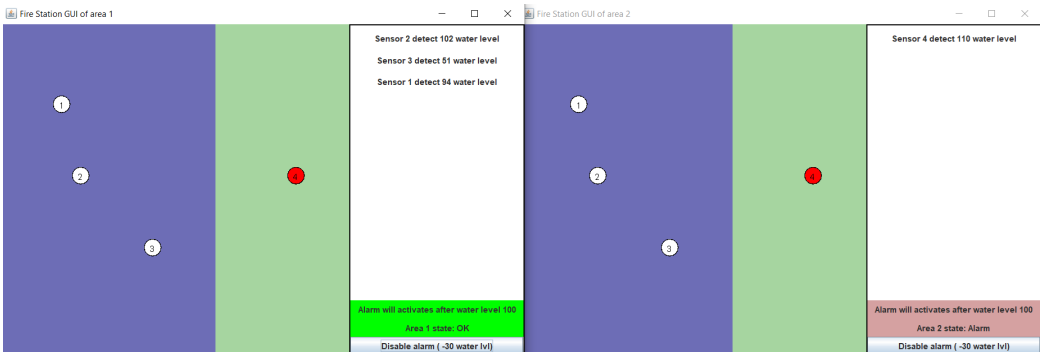

Figura 2.3: Due interfacce grafiche di due caserme dei pompieri. Si può notare che la seconda zona è in allarme mentre la prima non è, nonostante del fatto che un pluviometro della prima zona rileva la quantità d'acqua superiore della norma