Alessandro Pioggia

Assignment 1 - PCD

Deadline: 07/04/2024

CONTESTO DELLO STUDIO

Agent based system, in cui agenti agiscono sull'ambiente, in lettura (perceive) ed in scrittura (act), in seguito ad una fase di decide.

Agenti -> macchine

Environment -> strada/e

DESCRIZIONE STRATEGIA RISOLUTIVA

Si è optato per un approccio risolutivo incrementale, prendendo come riferimento le classi di testing, in ordine di complessità, ovvero:

- TrafficSimulationSingleRoadTwoCars: in questo caso si considera la necessità di gestire l'accesso ad un solo environment (Road) da parte di due agents (Cars), in modo sincronizzato;
- TrafficSimulationSeveralCars: rispetto alla classe precedente, aumenta il numero di Agenti;
- TrafficSimulationSingleRoadWithTrafficLight: alla simulazione viene aggiunto il semaforo;
- TrafficSimulationWithCrossRoads: oltre al semaforo, viene incluso anche un incrocio di due strade, è la classe definitiva.

Analisi

La gestione concorrente di 2 automobili, prevede la necessità che ogni singolo agente estenda un thread, in questo modo sarà possibile strutturare una policy per la suddivisione del lavoro, attraverso un'architettura concorrente. Eventuali problematiche:

- Accesso alle risorse condivise, le macchine devono accedere alle informazioni sull'environment, è necessario farlo in modo sincronizzato;
- Se tutti gli agenti vedo l'ambiente allo stesso modo, al tempo t,
- In generale l'accesso alle informazioni condivise deve essere gestito con mutex.

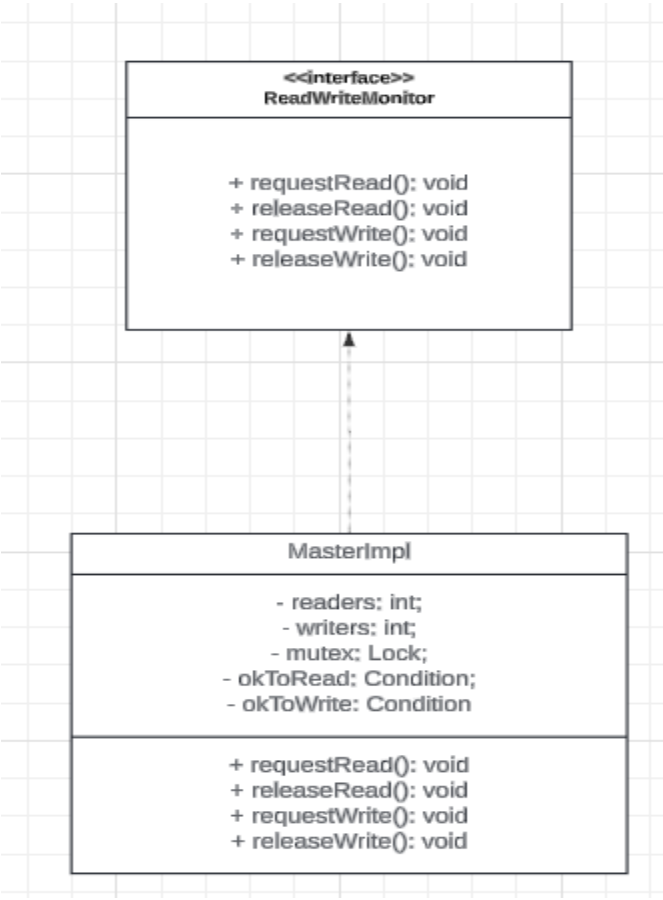
Readers & writers

Dall'analisi si può evincere che si tratta di un readers and writers problem, che comporta i seguenti vincoli.

In questo contesto, l'azione di read() è identificata dalla perceive(), mentre quella di write() dalla doAction(), che aggiorna l'environment.

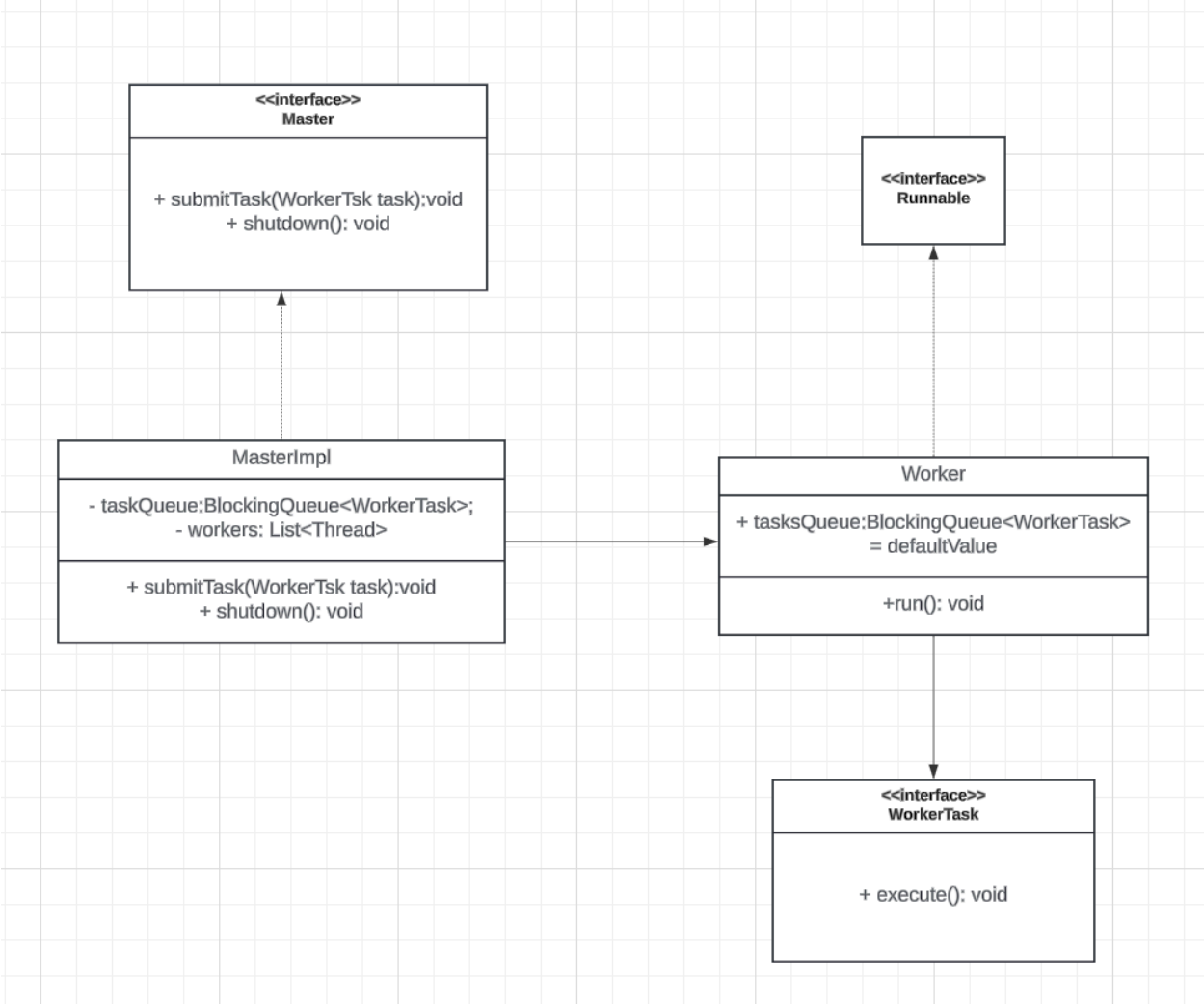
```
nReaders = getTotalReaders();
nWriters = getTotalWriters();

boolean getConstraints() {
  return nReaders >= 0 && nWriters = 0 && nWriters = 1 && nReaders = 0;
}

/* Readers and writers problem */
```

A livello implementativo, il readers-writers pattern, è stato realizzato attraverso l'implementazione di un monitor, generale, che prescinde dall'implementazione della simulazione.



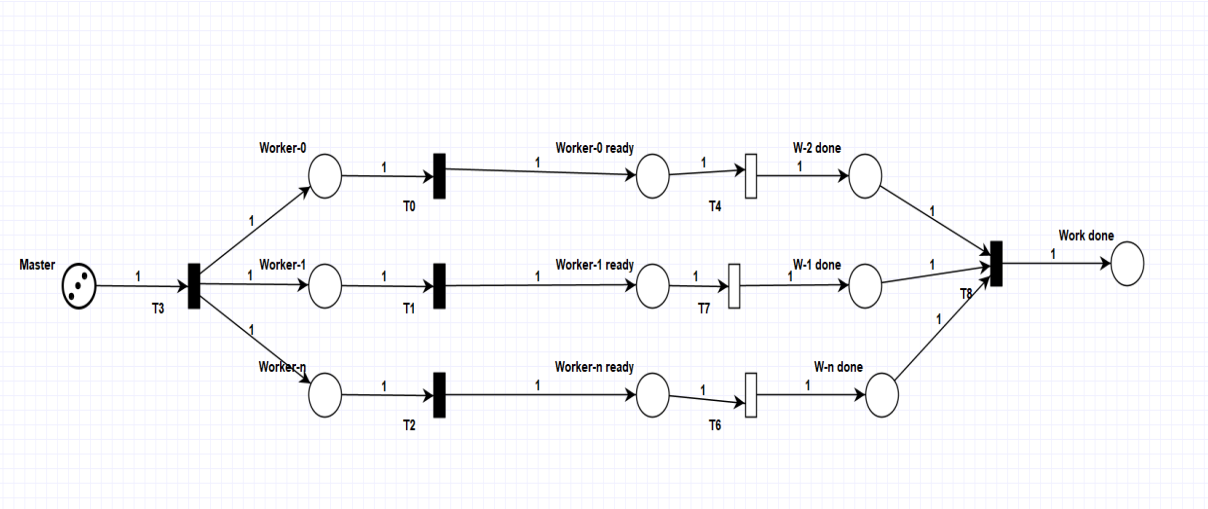
Master-worker architecture

Una volta stabilite le modalità di comunicazione nell'ambiente degli agenti, occorre strutturare una architettura concorrente, per fare in modo che vengano organizzati in modo corretto i task dei vari agenti. Relativamente all'assignment corrente, è stato scelto di applicare l'architettura Masters-workers, attraverso la realizzazione di un framework portabile e generale, utilizzabile anche in altri progetti.

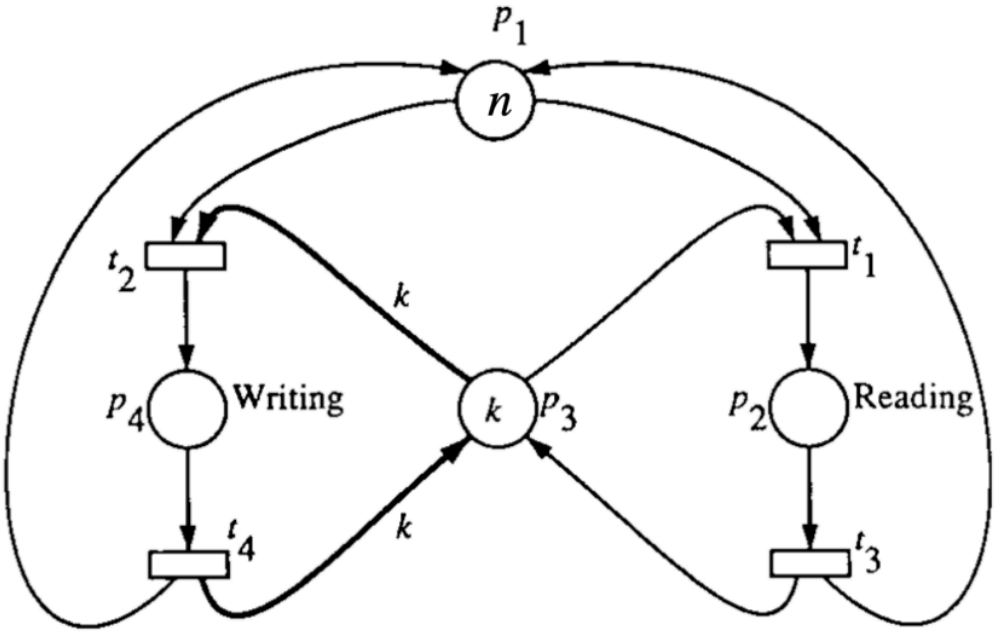


DESCRIZIONE DEL SISTEMA

Masters worker petri net



Readers-writers petri net



ANALISI PERFORMANCE

Per quanto riguarda le performance, è stato sfruttato il test RunTrafficSimulationMassiveTest.Java, con i parametri:

- numero di automobili: 10.000;
- numero di steps: 100.

Sono state effettuate diverse run e si è potuto evincere che il tempo di esecuzione della versione concorrente del simulatore assume questi valori:

- 10 thread in esecuzione -> 188 ms;
- 20 thread in esecuzione -> 161 ms;
- 30 thread in esecuzione -> 161 ms;
- 40+ thread in esecuzione -> 190ms.

Invece, effettuando alcune run sul programma sequenziale, si evince che in media, occorrono 16900 ms per terminare l'esecuzione.

A partire da questi parametri è stato possibile calcolare lo speedup, che nel caso migliore è del 105% circa.

Speedup = T sequenziale / T parallelo = 16.900 ms / 161 ms = 150.59