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.

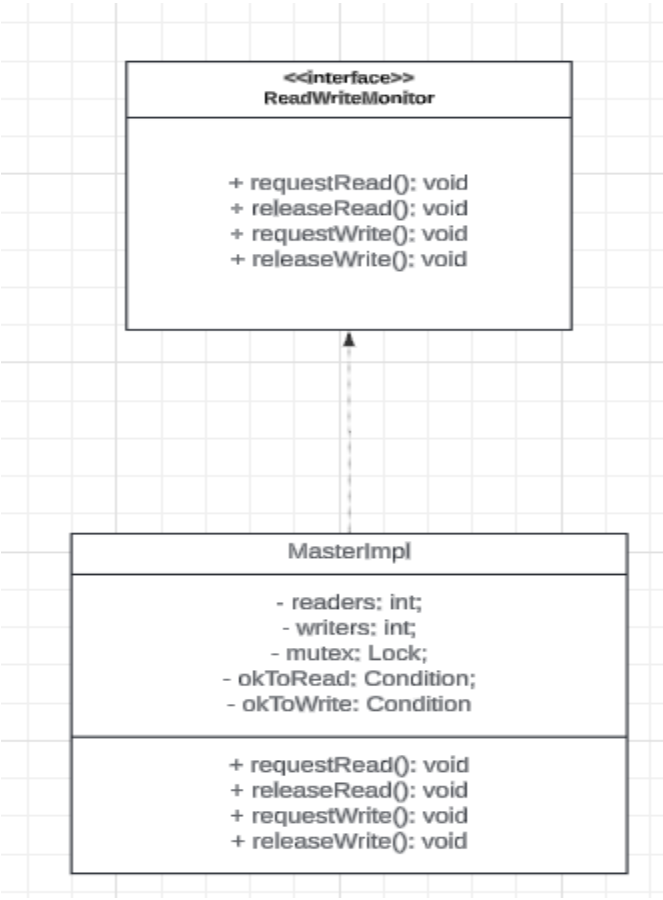
Readers & writers

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.

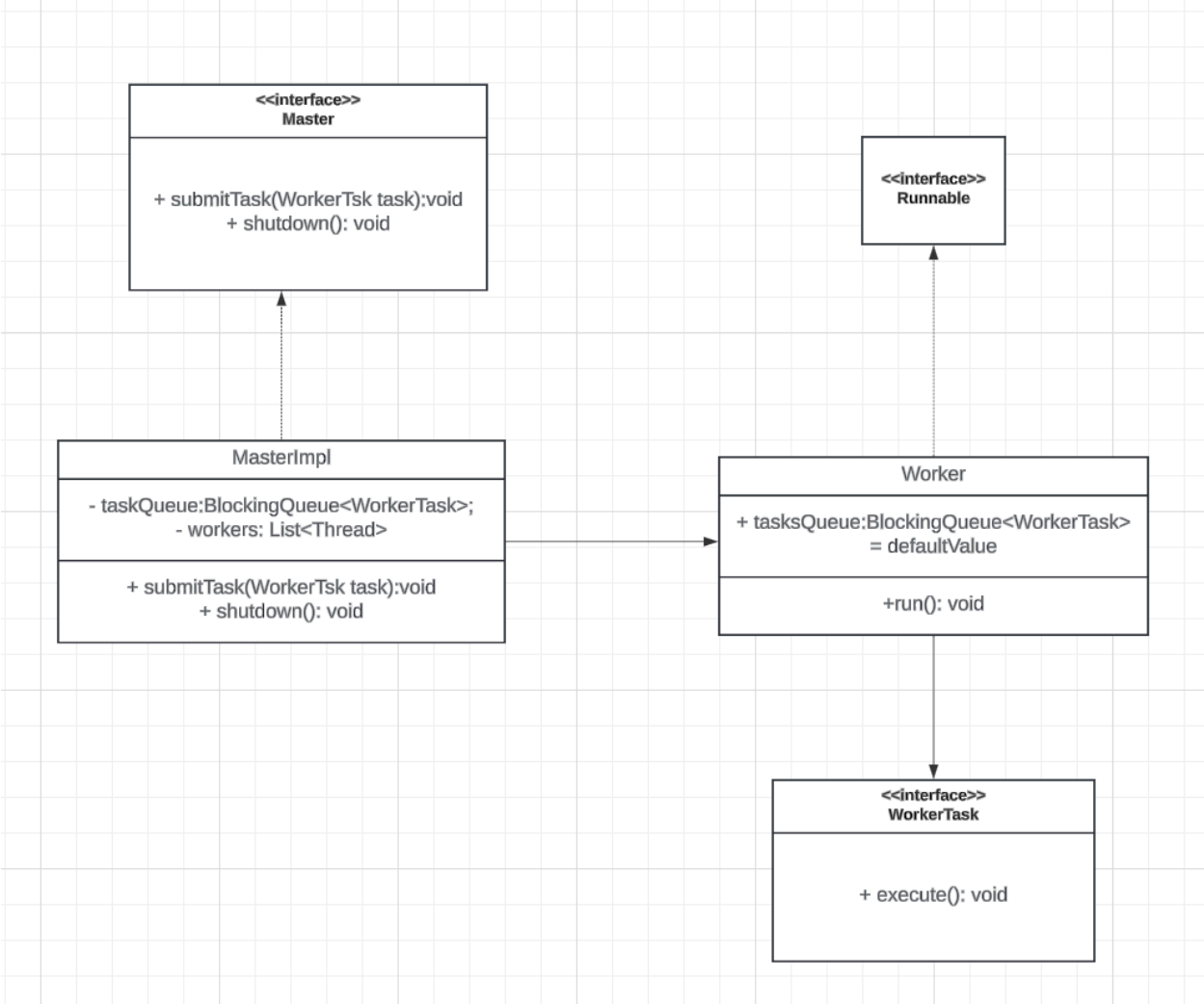
```
nReaders = getTotalReaders();
nWriters = getTotalWriters();

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.



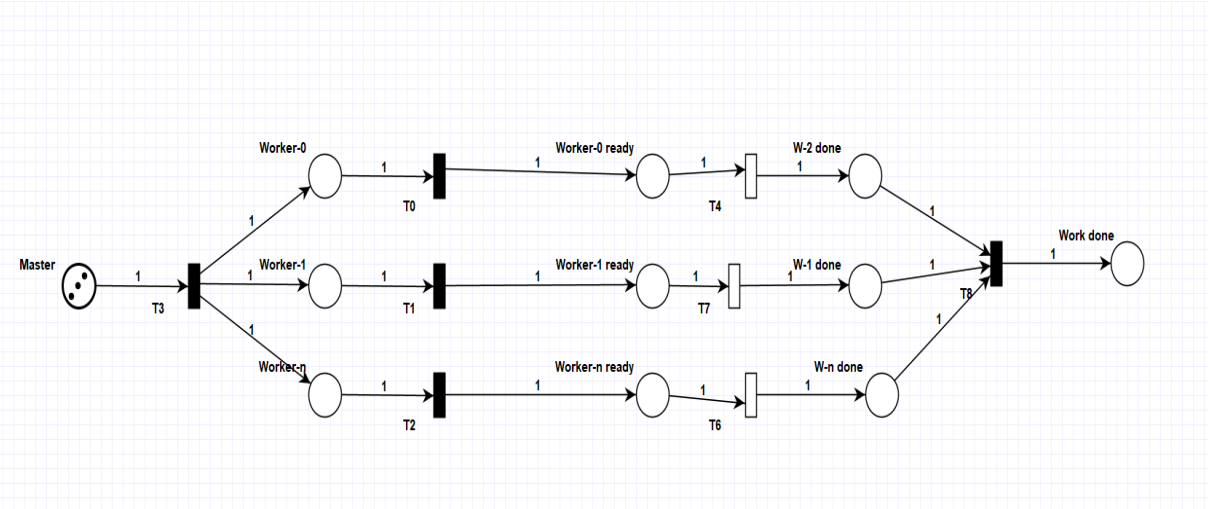
Master-worker architecture

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.

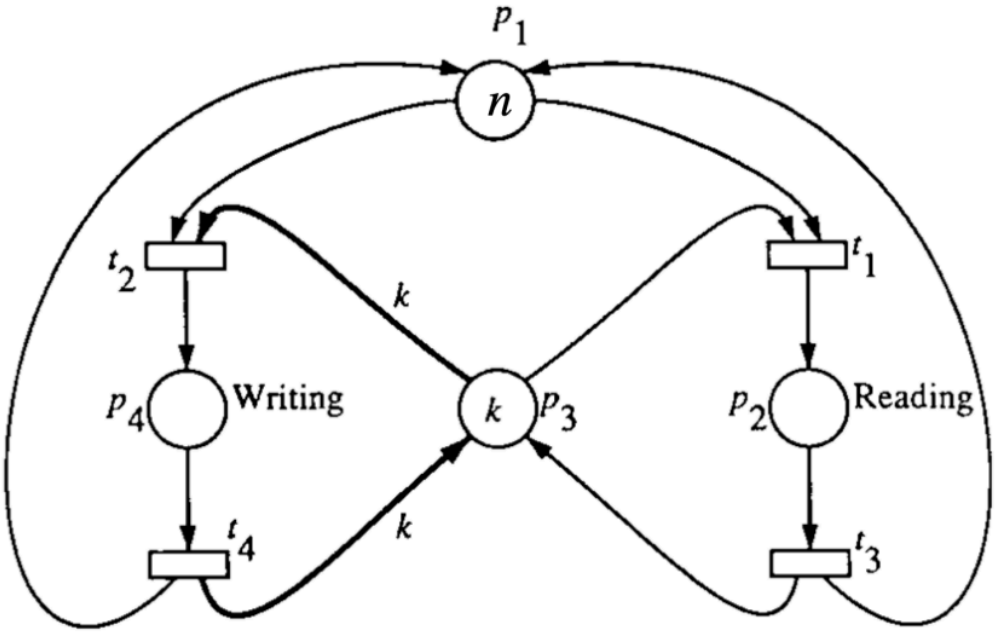


DESCRIZIONE DEL SISTEMA

Masters worker petri net



Readers-writers petri net



ANALISI PERFORMANCE

Calcolo speed-up

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

Calcolo speed-up (jpf)

Sono stati realizzati dei test, sfruttando il framework jpf, per verificare la correttezza del lavoro svolto, nelle pagine successive, è possibile vedere, come sono stati testati, separatamente l'architettura masters-workers e il readers writers problem (sfruttando il template jpf, non integrato nella consegna).

ReadWriteMonitorTest

Questo test, in breve, prevede la creazione di una simulazione, in cui n readers ed m writers, tentano di accedere ad un buffer condiviso (il lavoro viene simulato attraverso una sleep).

```
import gov.nasa.jpf.vm.Verify;
import pcd.ass01.monitors.ReadWriteMonitor;
import pcd.ass01.monitors.ReadWriteMonitorImpl;

public class ReadWriteMonitorTest {

    private static final int NUM_READERS = 3;
    private static final int NUM_WRITERS = 2;

    private static ReadWriteMonitor monitor;

    public static void main(String[] args) {
        monitor = new ReadWriteMonitorImpl();

        for (int i = 0; i < NUM_READERS; i++) {
            new Thread(new Reader(i)).start();
        }

        for (int i = 0; i < NUM_WRITERS; i++) {
            new Thread(new Writer(i)).start();
        }
    }

    static class Reader implements Runnable {
        private final int id;

        Reader(int id) {
            this.id = id;
        }

        @Override
        public void run() {
            while (true) {
                Verify.beginAtomic();
                monitor.requestRead();
                Verify.endAtomic();

                // Reading operation
                System.out.println("Reader " + id + " is reading...");

                Verify.beginAtomic();
                monitor.releaseRead();
                Verify.endAtomic();

                // Delay to simulate some work
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
        }
    }
```

```
static class Writer implements Runnable {
        private final int id;

        Writer(int id) {
            this.id = id;
        }

        @Override
        public void run() {
            while (true) {
                Verify.beginAtomic();
                monitor.requestWrite();
                Verify.endAtomic();

                // Writing operation
                System.out.println("Writer " + id + " is writing...");

                Verify.beginAtomic();
                monitor.releaseWrite();
                Verify.endAtomic();

                // Delay to simulate some work
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    Thread.currentThread().interrupt();
                }
            }
        }
    }
}
```

MasterWorker test

Il file jpf successivo invece, descrive un semplice test per verificare la correttezza dell'architettura masters-workers, verifica che il master istanzi correttamente i worker, che lavorano su thread separati.

```
import gov.nasa.jpf.vm.Verify;
import pcd.ass01.framework.Master;
import pcd.ass01.framework.MasterImpl;
import pcd.ass01.framework.WorkerTask;

public class MasterTest {

    private static final int NUM_TASKS = 5;

    private static Master master;

    public static void main(String[] args) {
        master = new MasterImpl();

        // Submit tasks to the master
        for (int i = 0; i < NUM_TASKS; i++) {
            master.submitTask(new TestTask(i));
        }

        // Shutdown the master
        master.shutdown();
    }

    static class TestTask implements WorkerTask {
        private final int id;

        TestTask(int id) {
            this.id = id;
        }

        @Override
        public void execute() {
            // Task execution logic
            System.out.println("Task " + id + " is being executed...");

            // Delay to simulate some work
            try {
                Thread.sleep(100);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
}
```