PCD ASS03 (1)

Federico Bravetti
federico.bravetti2@studio.unibo.it

Tommaso Patriti
tommaso.patriti@studio.unibo.it

Alex Testa
alex.testa@studio.unibo.it

I. Analisi del problema

Il progetto in questione consiste nello sviluppo del simulatore descritto nell'Assigment1 adottando un'architettura ad attori utilizzando il framework *Akka*.

A. Architettura del sistema

Prima di tutto è stata analizzata l'archiettura iniziale per semplificarla ed adattarla meglio ad una soluzione ad attori. L'architettura risultante è riportata in Fig. 1.

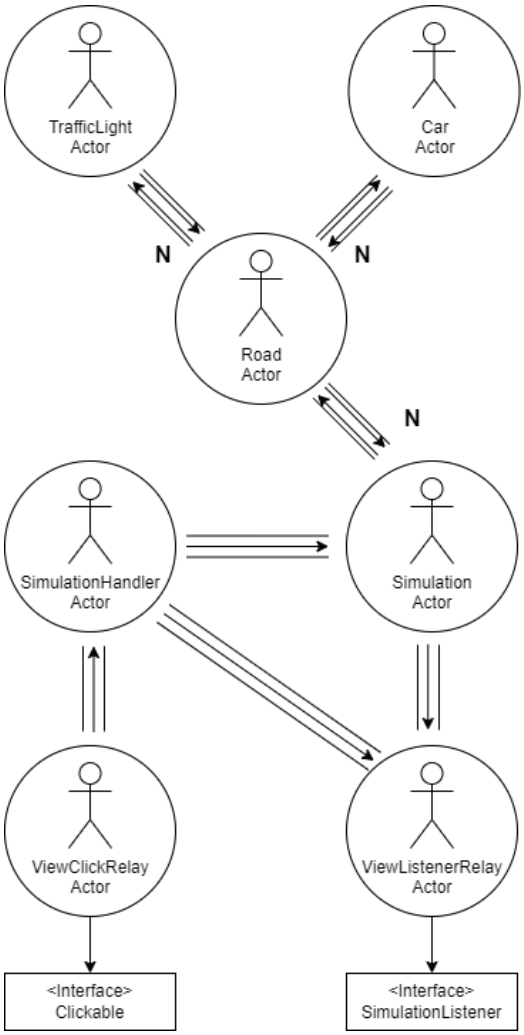

Figure 1. Architettura

Poiché nel progetto iniziale non venivano considerate interazioni tra entità associate a Road diverse si è deciso di rimuoverere il concetto di Environment andando a delegare le rispettive funzionalità alle Road.

Da un punto di vista gerarchico gli attori sono organizzati secondo lo schema in Fig. 2

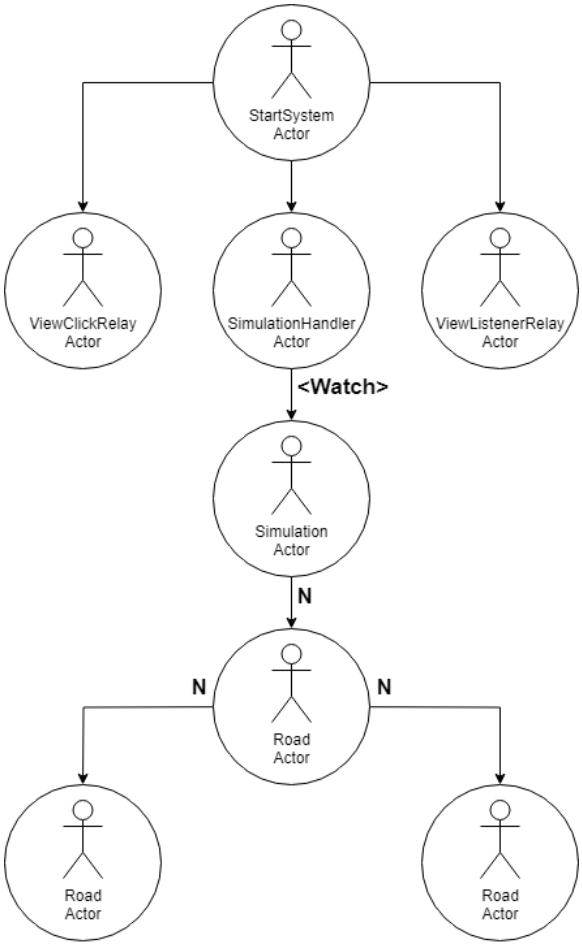

Figure 2. Gerarchia Attori

Per ottimizzare l'allocazione delle risorse al termine della simulazione il *SimulationActor* (e il relativo sotto albero) viene terminato. Per gestire correttamente la terminazione il SimulationActor è osservato dal padre *SimulationHandlerActor*, così che potrà gestire l'evento di terminazione.

II. Comportamento del Sistema

Di seguito sarà analizzato il comportamento del sistema sia andando a considerare le interazioni es-

terne tra attori sia considerando la loro evoluzione interna in termini di *Behavior*.

A. Interazione tra Attori

In primo luogo si vuole analizzare l'evoluzione temporale della simulazione andando a considerare i messaggi scambiati tra i vari attori. Il comportamento essenziale del sistema è definito dallo scambio di messaggi riportato in Fig. 3, a cui è associato lo schema riportato in Fig. 4 che rappresenta l'esecuzione di uno step.

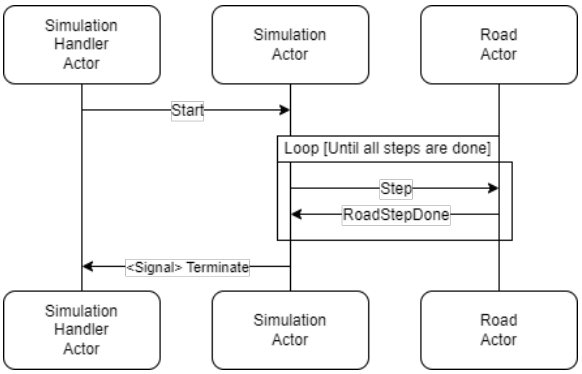

Figure 3. Evoluzione della simulazione

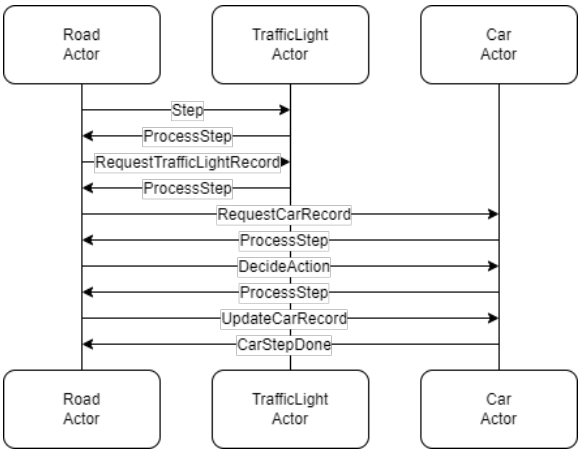

Figure 4. Esecuzione di uno step

Entrambi questi schemi sono semplificati, infatti in generale:

- Una Simulation può interessare più Road
- Una Road può essere composta da più TrafficLight
- Una Road può essere composta da più Car

Per gestire correttamente le interazioni nei casi appena elencati è necessario introdurre un attore ausiliario in grado di raggruppare le risposte delle diverse entità prima di inviare una risposta complessiva al mittente. Questo attore è stato chiamato *Aggregator*.

Un esempio dell'utilizzo di quest'ultimo è riportato in Fig. 5, in questo caso è utilizzato da SimulationActor per comunicare con i diversi *RoadActor* ma in modo analogo è utilizzato anche dai RoadActor per comunicare con i vari *TrafficLightActor* e i vari *CarActor*.

In particolare ogni volta che è necessario interagire con un gruppo di entità viene effettuato uno spawn di questo attore, che sarà terminato non appena avrà inviato la risposta aggregata al mittente.

Inoltre si può notare che il SimulationActor ha un peculiare comportamento di auto-trigger. Infatti dopo aver ricevuto il messaggio di *Start* si auto-invia un messaggio di *Step* e, durante l'esecuzione di questo, programma l'Aggregator per l'invio del successivo messaggio di Step, così facendo è in grado di avanzare autonomamente fino al termine della simulazione.

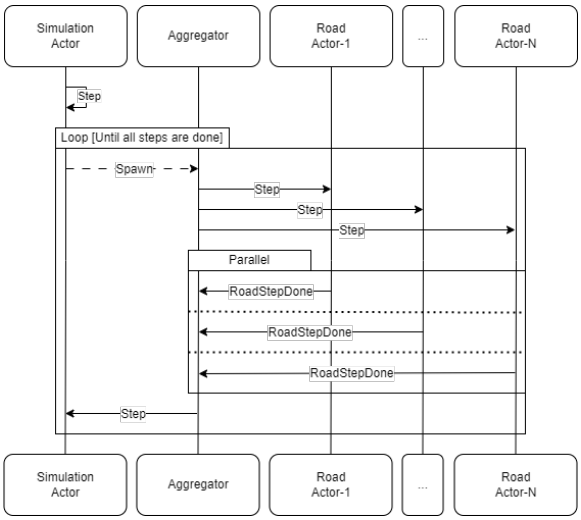

Figure 5. Evoluzione dettagliata della simulazione

Per tutto quello che riguarda l'esecuzione della simulazione è necessario eseguire i diversi passaggi in un ordine preciso. Questo, come si può notare anche dai Sequence Diagram, ha reso necessario l'utilizzo di un protocollo di tipo Request-Response. Per quanto rigurda l'apertura e la chiusura della GUI della simulazione per semplicità si è scelta una QoS di tipo Fire-And-Forget.

NOTA: l'Aggregator in questo caso è richiesto poiché serve a coordinare l'avanzamento della simulazione definendo punti di sincronizzazione. In generale non è necessario nelle comunicazioni one-to-many.

B. Attori come DFA

Oltre a interagire tra di loro alcuni attori modificano il loro Behavior a seconda dello stato dell'interazione in cui si trovano. Per rappresentare questa evoluzione sono stati utilizzati dei DFA, di seguito sono rappresentati quello di SimulationHandlerActor (Fig. 6), di SimulationActor (Fig. 7) e di RoadActor (Fig. 8). Tutti gli altri attori mantengono

lo stesso comportamento dall'inizio alla fine del programma.

In generale in ogni stato ogni attore è in grado di ricevere tutti i messaggi ad esso associati, quindi a quelli non riportati negli automi è stato associato il comportamento *Behaviors.unhandled*.

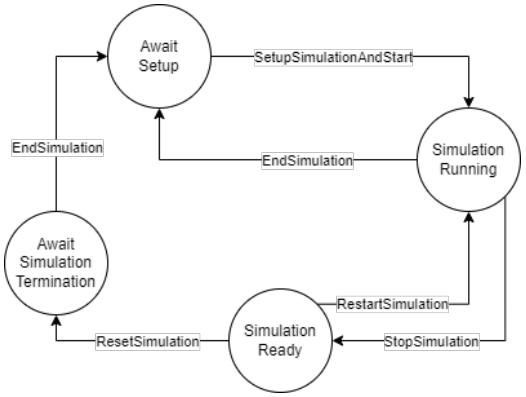

Figure 6. SimulationHandlerActor DFA

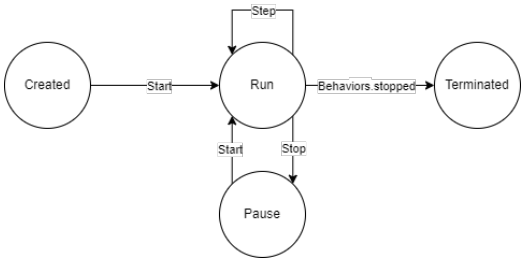

Figure 7. SimulationActor DFA

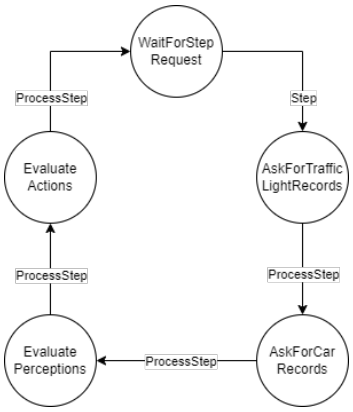

Figure 8. RoadActor DFA

NOTA: Ogni messaggio di Stop che riceve l'attore Simulation nello stato Run è seguito da un messaggio di Step (dato che è stato programmato durante l'esecuzione dello step precedente) per cui si è optato per l'utilizzo di uno stash con buffer unitario per la gestione di questo messaggio nello stato Pause.

III. Test

Per valutare la correttezza dei risultati ottenuti dalle simulazioni sono stati implementati test analoghi a quelli dell'Assigment1.

IV. Performance

Rispetto all'implementazione multi-thread in java, questa implementazione ha un decremento di performance di circa 12 volte. Evidentemente, nonostante sia ottimizzata, la comunicazione ad attori su JVM risulta drasticamente più inefficiente.