RELAZIONE ISS25 - FASE 2

Marco Crisafulli

Nella prima fase abbiamo esplorato e sperimentato con i vari protocolli di comunicazione (MQTT, TCP, Socket) per implementare il gioco Conway Life.

Verso uno sviluppo Top-Down

Adesso spostiamo la nostra attenzione su un nuovo problema:
Passare da uno sviluppo di tipo naive bottom up ad uno più strutturato di tipo Top Down.
Uno sviluppo Top Down prevede un'accurata analisi del problema e dei requisiti per colmare l'abstraction gap.
Abstraction gap che viene definito così nel materiale didattico:
Con il termine abstraction gap intendiamo denotare la **distanza** tra le mosse di base fornite da un automa o da un linguaggio di programmazione (general-purpose) e quelle necessarie per affrontare in modo adeguato un problema applicativo.
Grazie ad uno sviluppo Top Down possiamo attuare le migliori strategie per realizzare un software efficace, progettando e scegliendo le tecnologie piu' adeguate o sviluppandole in caso di necessità.
Chiaramente è necessario allocare una quantità di tempo sufficiente alle fasi di analisi e progettazione e non buttarsi a scrivere codice di getto.

QakActors25 e actors

Cos'è QaK? Allego la definizione presente sul materiale:
La Q/q nella parola *QActor*, significa "quasi" poiché il linguaggio non è inteso come un linguaggio di programmazione generico, ma piuttosto un linguaggio di modellazione eseguibile, da utilizzare durante l'analisi del problema e il progetto di protitpi di sistemi distribuiti, i cui componenti sono attori che si comportano come un Automa a stati finiti, in stretta relazione con l'idea di sistemi basati su Microservizi. La K finale sta per Kotlin, che serve ad evitare l'uso di Akka.

Un attore qak specializza la classe astratta it.unibo.kactor.ActorBasicFsm.kt che a sua volta specializza la classe astratta it.unibo.kactor.ActorBasic.kt, entrambe definite nella Qak infrastructure.

L'uso di questi attori va a semplificare il processo di sviluppo perché ci induce a ragionare e modellare il problema con un automa a stati finiti (Sono relativamente intuitivi e hanno delle regole ben definite rispetto ad una generica Macchina di Turing).

Riporto nuovamente il materiale:
Il Linguaggio qak reso disponibile dalla Qak software factory intende fornire un linguaggio per la definizione di modelli eseguibili di un sistema, basati su un insieme di `concetti volti a catturare l'idea` che un sistema software (distribuito) possiede le seguenti caratteristiche:

- il sistema è formato da una insieme di attori
- gli attori interagiscono scambiandosi messaggi (di tipo IApplMessage, nel nostro caso)
- un attore è un ente autonomo, capace di elaborare messaggi. Pertanto la struttura del codice di un attore si presta ad essere modellata come un Automi a stati finiti
- gli attori sono raggruppati in contesti che li abilitano a interazioni via rete
- i contesti possono essere allocati (deployed) su uno o più nodi computazionali, fisici o virtuali

Applicazione realizzate con Qak

Abbiamo visto e commentato a lezione varie versioni del gioco ConwayLife realizzate mediante l'uso degli actors. Prima un attore come controller, poi come cella.
Ed è qui che è emersa la necessità di un'ulteriore analisi del problema.
Le domande poste sono: Serve un orchestratore? Come faccio ad identificare ogni cella? Come fa una cella a comunicare e conoscere lo stato delle celle vicine?
Andiamo con ordine:
Serve un Orchestratore o no?
Sicuramente si potrebbe progettare un sistema senza un orchestratore, ma ciò andrebbe a complicare lo sviluppo e ridurre la tolleranza a guasti o banalmente introdurre (e quindi pensare e progettare) un nuovo sistema per iniziare la partita e per gestire le epoche.
Con un orchestratore possiamo gestire tutto in modo più snello ed esplicito.

La partita inizia quando sono connesse tutte le celle e ad ogni epoca si attende che tutte le celle abbiano cambiato stato o no e quindi comunicato all'orchestratore che sono pronte.

Come faccio ad identificare ogni cella?

Siamo in presenza di un sistema distribuito dove ogni cella gira su un nodo suo, perciò all'avvio ogni cella avrà un nome proprio scelto dall'utente o random.

Quando la cella si connette all'orchestratore allora le verrà assegnato un nome corrispondente alla sua coordinata nella griglia ed uguale al nome della topic.

Ogni cella conosce la dimensione della griglia e la propria coordinata ed è perciò in grado di sapere chi e quanti sono i propri vicini (3,5 o 8), grazie a questo nome sarà anche in grado di ricavare le topic MQTT sulle quali trasmettere.

Come fa una cella a comunicare e conoscere lo stato delle celle vicine?

Ogni cella grazie al proprio nome (corrispondente alla topic) conosce il topic dei vicini.

Una cella comunica il proprio stato a tutti i vicini ogni epoca in modo tale da evolvere grazie a questa informazione e infine comunicare all'orchestratore di aver terminato la computazione relativa all'epoca.

Come fa però a connettersi al configuratore di gioco? Sfruttiamo il servizio Eureka introdotto a lezione che funge da registro in cui i servizi possono registrarsi e individuare altri servizi registrati. Questo servizio verrà lanciato con Docker.

Quindi la cella è un semplice oggetto o un attore? (in questo sistema)

Attore perché sei su una macchina, devi rapportarti al mondo esterno e agire in base alle informazioni.

Raspberry Pi

Il lavoro di analisi del problema svolto sopra è mirato ad un'implementazione del gioco conway life su dei nodi indipendenti (dei raspberry pi) ognuno con un led che indica lo stato della cella corrispondente.

Abbiamo visto anche altri problemi da modellare e portare su un dispositivo raspberry (Sonar) che integrano sensori esterni da cui il dispositivo ricava informazioni sull'ambiente e sulla base di queste informazioni esegue dei calcoli e cambia il proprio stato. (operazioni che lo rendono un attore)

Qui definiamo i Situated Actors (Agenti Software Situati):
Entità che percepiscono il loro ambiente (dinamico), agiscono sulla base di informazioni ricavate dall'ambiente, sono in grado di modificare l'ambiente e il loro comportamento non è determinato unicamente da regole pre-programmate, ma anche dal loro contesto attuale e dalla storia delle interazioni all'interno dell'ambiente.

Ci avviamo così all'introduzione dei concetti di Robot Reali e Simulati che affronteremo nella Fase 3.