RELAZIONE FINE FASE2-ISS25

DI SAVIGNANO SIRIA

Repository Git: https://github.com/sirius-22/iss25SavignanoSiria

DAI MICROSERVIZI AGLI ATTORI

RIASSUNTO

In questa fase abbiamo evoluto la soluzione della fase 1, progettando e realizzando una versione **distribuita** del Gioco della Vita di Conway tramite il **linguaggio DSL Qak** e il **modello ad attori**.
Ogni cella è rappresentata da un **attore autonomo**, in grado di comunicare con i propri vicini per determinare il proprio stato futuro, secondo le regole del gioco.

Per affrontare le sfide della sincronizzazione e del coordinamento, abbiamo introdotto un attore speciale, il **gameMaster**, che assume il ruolo di **orchestratore** del sistema: detta i tempi di evoluzione, avvia e interrompe le generazioni, e permette alle celle di operare in modo distribuito ma coordinato.
Questo approccio, detto **orchestrato**, supera i limiti di una soluzione puramente **coreografata**, in cui ogni attore opera in completa autonomia rischiando inconsistenze e ritardi non controllabili, oltre ad un rilevante incremento dei **costi di analisi e implementazione** di una soluzione del genere.

Il linguaggio **Qak** si è rivelato fondamentale nel supportare questo paradigma: attraverso costrutti ad alto livello e una semantica ben definita, ci ha permesso di descrivere il comportamento degli attori come **automi a stati finiti**. Ogni attore è naturalmente predisposto alla **comunicazione asincrona**, e grazie all'infrastruttura sottostante non è necessario gestire esplicitamente né invii né ricezioni di messaggi.

PERCHÉ PROPRIO IL GIOCO DELLA VITA DI CONWAY?

Il **Gioco della Vita di Conway** continua a essere il modello ideale per l'evoluzione progettuale perché naturale terreno di sperimentazione per l'ingegneria dei **sistemi distribuiti.** La transizione dalla versione oggetto-centrica a quella basata su attori è resa più semplice dal **comportamento locale e deterministico** delle celle: ogni cella conosce il proprio stato e quello dei vicini, e può decidere autonomamente la propria evoluzione.
Questo si sposa perfettamente con il **modello ad attori**, in cui **ogni cella diventa un attore** autonomo, in grado di ricevere messaggi, elaborare in parallelo e inviare notifiche.
Inoltre, lo sviluppo con attori supporta il **design for change**, facilitando l'estensione e la distribuzione del sistema.

FINALITÀ E SCOPO

In questa seconda fase, l'obiettivo è modellare **sistemi distribuiti basati su microservizi**, trattando ogni microservizio come un **attore indipendente**, in linea con il **modello ad attori.**
L'approccio scelto si fonda sull'idea che un attore possa rappresentare un **servizio autonomo**, dotato di stato locale e logica propria, in grado di comunicare con altri servizi solo tramite **messaggi asincroni**.

L'utilizzo degli attori non sostituisce l'architettura a microservizi, ma ne offre un'alternativa implementativa più orientata alla modellazione comportamentale e alla **gestione concorrente**.

Ciò permette di esplorare **i benefici del paradigma ad attori** nel progettare un sistema **modulare**, **distribuito** e facilmente **scalabile**.

COMPETENZE ACQUISITE

Durante questa fase si è avuta l'opportunità di approfondire e sperimentare l'uso del **linguaggio Qak (Quasi-Actor-Kotlin),** fornito dal docente, pensato per la modellazione eseguibile di sistemi distribuiti.

Si è osservato come, in Qak, ogni attore **(QActor**) sia **intrinsecamente capace di comunicare**: la gestione dei messaggi, la comunicazione asincrona, e la struttura a code FIFO sono geneticamente integrate nella semantica del modello, senza necessità di ulteriori astrazioni infrastrutturali. L'interazione tra attori avviene attraverso **messaggi** (dispatch/event), con l'infrastruttura che si occupa di ricezione, buffering e consegna nel rispetto del **protocollo di comunicazione** selezionato (es. TCP, MQTT, CoAP).

In un secondo momento si è potuto supportare la configurazione di un **Raspberry Py** in modalità headless, con lo scopo, nella prossima fase, di poter concretizzare la logica ad attori applicandola al progetto conwayLife.

KEY POINTS

In questa seconda fase, sono emerse alcune buone pratiche e pattern fondamentali dell'ingegneria del software:

- Single Responsibility per attore
- Design for change
- Modello ad attori
- Modularità
- Disaccoppiamento tra logica e infrastruttura
- Automa a stati finiti
- Message-driven architecture

IL RUOLO DEL LINGUAGGIO

Il linguaggio di programmazione utilizzato, rende disponibile il proprio **spazio concettuale**, ovvero la capacità espressiva del linguaggio stesso. Utilizzare un linguaggio di programmazione di alto livello, come java in questo caso, permette l'**abstraction gap**: la possibilità di nascondere i dettagli poco rilevanti, come l'hardware sottostante, per potersi concentrare sugli aspetti ritenuti interessanti.

Il linguaggio Qak nasce con l'obiettivo di facilitare l'implementazione di architetture a **microservizi**, basandosi su un **modello ad attori** il cui comportamento è descritto mediante **automi a stati finiti di tipo Moore**. Questo DSL ha svolto un ruolo centrale nel colmare l'**abstraction gap** tra modello e implementazione. Grazie alla sua sintassi orientata agli **attori**, ha permesso di descrivere **comportamenti e interazioni** in modo ad alto livello, lasciando a Qak l'onere di generare la struttura sottostante. Si sono potute sfruttare le sue **potenzialità semantiche e strutturali**, per merito della tecnologia **Xtext**. Questa, infatti, permette di definire in modo semplice una **Domain-Specific-Language**, generando automaticamente la sintassi e offrendo strumenti per associare a ciascun costrutto una semantica comportamentale, attraverso trasformazioni ad hoc nel back-end. Questi fattori hanno reso possibile focalizzarsi sulla **logica del sistema**, ignorando i dettagli implementativi legati alla concorrenza e alla comunicazione. Il linguaggio, con i suoi costrutti nativi per gli attori, i messaggi, e le transizioni, ha consentito di **astrarre i meccanismi di basso livello**, favorendo la chiarezza, la riusabilità e **l'evoluzione incrementale** del sistema.