RELAZIONE FINE FASE1-ISS25

DI SAVIGNANO SIRIA

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

DAGLI OGGETTI AI MICROSERVIZI

RIASSUNTO

L'applicazione ConwayLife25 è un'applicazione Java che simula l'evoluzione di un insieme di celle in un ambiente bidimensionale.

Per prima cosa si è progettato il core business (**software driven development**), con output inizialmente stampato a console.

Il modello è stato progettato in maniera modulare per sfruttare il **principio di singola responsabilità**. Si sono a tal scopo introdotte le classi Grid e Cell e si è separata la parte di logica applicativa da quella di visualizzazione dell'output. In seguito, è stata aggiunta la parte di **testing**, parte essenziale dello sviluppo di qualsiasi applicazione. In questo caso si computato il comportamento che avrebbero avuto le celle ad ogni epoca in simulazioni di cui si conosceva a priori il risultato.

L'attenzione è stata successivamente dirottata sulla progettazione di una GUI, un dispositivo avanzato di input-output, per l'applicazione, di modo da renderla appetibile per l'utente finale.

Abbiamo utilizzato il **framework spring boot** per l'organizzazione dei package e per la **Dependency Injection.**

La business logic può essere integrata all'interno dell'applicazione in 2 modi diversi: innestandola direttamente all'interno del progetto o, in maniera implicita, creando un progetto a parte e integrandolo come dipendenza (ad esempio un jar).

Inizialmente, in conwaygui, si è optato per la prima soluzione, in quanto l'applicazione può ancora essere soggetta a cambiamenti e variazioni (**design for change**).

Si è deciso di adottare il modello **Model-View-Controller**, dove il model è il core business dell'applicazione, la view è la pagina html e il controller è il controller di Spring, il quale richiama il controller dell'applicazione.

La comunicazione, **full duplex e asincrona**, tra frontend e backend è gestita tramite websocket. Questo approccio ha reso possibile la machine to machine interaction, realizzata tramite un programma, chiamato caller, che invia i comandi da dare a ConwayLife25 tramite websocket.

A tal fine è stata sviluppata la classe ConwayCallerWSInteraction, che utilizza una WsConnection creata tramite ConnectionFactory per inviare e ricevere messaggi senza esporre i dettagli della libreria javax.websocket (**Tecnology independency**). La connessione è **osservabile**: i messaggi emessi dal servizio vengono automaticamente notificati agli oggetti osservatori attraverso il metodo update, senza intervento esplicito del programma.

Parallelamente, il sistema è stato evoluto per utilizzare **MQTT** come meccanismo di comunicazione tra la GUI e il servizio logico, introducendo astrazioni di comunicazione più avanzate rispetto all'uso diretto della libreria Paho. Il progetto conway25JavaMqtt utilizza un dispositivo di output OutInMqtt

per gestire lo scambio di messaggi con la GUI, inviando aggiornamenti sulle celle e ricevendo comandi tramite **topic** specifici.

La GUI stessa, nel progetto conwayguialone, è stata trasformata in un **microservizio indipendente** che comunica con il core via MQTT. Tuttavia, sono state esplorate ottimizzazioni alternative basate su WebSocket (OutWs) per la trasmissione delle informazioni verso la GUI, pur evidenziando come ciò possa creare dipendenze indesiderate tra componenti.

Infine, è stata considerata un'ottimizzazione lato MQTT per ridurre il traffico, raggruppando i messaggi di aggiornamento in blocco, sebbene ciò contrasti con l'obiettivo ultimo di una distribuzione delle celle su più nodi computazionali.

PERCHÉ PROPRIO IL GIOCO DELLA VITA DI CONWAY?

Il Gioco della Vita è stato scelto come applicazione di riferimento perché rappresenta in modo perfetto un **Sistema Complesso**, ma può essere costruito e compreso passo dopo passo come un **Sistema Complicato**. Infatti, ha regole semplici e facilmente programmabili (sistema complicato), tuttavia le sue dinamiche evolutive portano a comportamenti imprevedibili, emergenti e non riducibili a singole regole (sistema complesso).

FINALITÀ E SCOPO

Le finalità di questa fase sono quelle di progettare e sviluppare il gioco Conway Life partendo da linguaggi noti come JavaScript e Java, seguendo un approccio bottom-up. L'obiettivo principale è costruire un sistema modulare ed estensibile, pensato per evolvere progressivamente senza modificare quanto già realizzato e testato, in linea con i principi di buona progettazione software. Si vuole inoltre realizzare un'interfaccia HTML in grado di comunicare via WebSocket con il backend, trasformare il sistema in un microservizio autonomo usando SpringBoot e gestire la comunicazione con la GUI tramite MQTT. Infine, il sistema deve essere distribuito come immagine Docker e la GUI deve diventare un microservizio indipendente, in grado di comunicare tramite astrazioni di comunicazione definite ad hoc.

Nelle prossime fasi l'obiettivo principale sarà trasformare il sistema in una struttura sempre più flessibile e scalabile, capace di evolvere da una semplice applicazione a un sistema distribuito basato su attori. Si vuole abbandonare architetture rigide per adottare un modello in cui ogni componente può comunicare attraverso messaggi in modo autonomo. Per supportare questo passaggio, verrà utilizzato un linguaggio di modellazione dedicato (qak) che consente di descrivere il comportamento e le interazioni degli attori, semplificando l'evoluzione del sistema. Questo approccio permetterà di astrarre i dettagli implementativi, concentrandosi sulle relazioni ad alto livello e agevolando il refactoring continuo, fino a distribuire le singole celle su più dispositivi fisici in un ambiente realmente distribuito.

REALIZZAZIONE E SPERIMENTAZIONE

In questa prima fase si è sperimentato il sistema ConwayLife25 di cui si è fatto il refactoring, apportando modifiche come la separazione tra la logica di rappresentazione e quella di gioco.

Si è ritenuta di particolare interesse l'organizzazione dei package e la dependency injection ottenuta tramite il framework springboot, in particolare spring initializr.

Si è sperimentato la containerizzazione tramite Docker, distribuendo il sistema come immagine portabile.

In maniera trasversale si è presa confidenza con l'utilizzo di git e l'organizzazione delle versioni del progetto, dando particolare attenzione alla documentazione dei passaggi svolti.

COMPETENZE ACQUISITE

Durante la prima fase del progetto sono state sviluppate competenze sia tecnologiche sia concettuali, che costituiscono la base per affrontare le successive evoluzioni del sistema.

Dal punto di vista tecnologico, si è lavorato con linguaggi noti come JavaScript e Java, seguendo un approccio **bottom-up** per la realizzazione del Gioco della Vita di Conway. È stata progettata e realizzata una pagina HTML capace di fungere da dispositivo di input-output evoluto, interagendo con un server tramite **WebSocket**, migliorando la capacità di integrare client e server in tempo reale.

Si è appreso l'utilizzo di **Gradle** come strumento di build e gestione delle dipendenze, imparando a configurare e organizzare il progetto in modo modulare ed efficiente. Inoltre, si è acquisita dimestichezza con l'ambiente di sviluppo **Eclipse**, sfruttandolo per la scrittura, il debug e la gestione dei progetti software strutturati.

Successivamente, si è imparato a trasformare il sistema in un **microservizio autonomo**, utilizzando **SpringBoot**, rendendolo scalabile e pronto a dialogare con altri componenti attraverso **MQTT**, un protocollo leggero e adatto alla comunicazione distribuita. La distribuzione del sistema come immagine **Docker** ha permesso di acquisire familiarità con la containerizzazione, semplificando il deploy e l'utilizzo su qualsiasi macchina, conoscendone solo la "vista esterna".

Dal punto di vista concettuale, questa fase ha permesso di approfondire l'importanza di progettare sistemi evolutivi, in grado di crescere senza modificare le funzionalità già testate, seguendo i **principi di modularità e riusabilità**. Si è lavorato sul concetto di **microservizio** come unità indipendente e si è compresa la necessità di utilizzare **astrazioni di comunicazione**, capaci di nascondere i dettagli implementativi e facilitare l'integrazione tra componenti diversi. Inoltre, si è rafforzata la capacità di pensare in termini di **architetture distribuite**, considerando la GUI come un dispositivo I/O indipendente, in comunicazione continua con il sistema applicativo.

MOTIVAZIONI ALLA BASE DI JAVA

Java come linguaggio di programmazione adottato è un'ottima scelta in quanto, essendo un linguaggio consolidato e maturo, offre un'ampia gamma di librerie. Oltretutto è conosciuto da tutti gli studenti del corso, quindi è didatticamente adeguato. SpringBoot è una scelta comune per creare microservizi, grazie alla sua capacità di semplificare la configurazione e l'integrazione di componenti, oltre a rendere più efficienti le operazioni di gestione e distribuzione.

Ciò non toglie che alternative moderne, come node.js o go, potrebbero risultare più adatte in termini di prestazioni o facilità di sviluppo, oltre ad essere qualcosa di nuovo e interessante da imparare.

KEY POINTS

I key points utilizzati nella prima fase sono i seguenti:

- Analisi Bottom up
- Singola responsabilità

- Design for change
- Separazione tra logica e presentazione
- Dependency Injection
- Software Driven Development
- Model-View-Controller
- Design for change
- Abstraction gap
- Pattern observer
- Tecnology independency

LE LIBRERIE CUSTOM

Il ruolo dello sviluppo di librerie 'custom' durante la realizzazione di un sistema software è quello di fornire strumenti di comunicazione riutilizzabili, modulari e indipendenti dai dettagli implementativi specifici di un singolo progetto. Queste librerie astraggono i meccanismi di basso livello (come socket, connessioni di rete, gestione dei protocolli) e mettono a disposizione interfacce e classi che permettono di interagire attraverso protocolli diversi in modo uniforme e trasparente.

Nel caso specifico del progetto, la libreria unibo.basicomm23-1.0.jar ha introdotto il concetto di interconnessione, permettendo di gestire diversi canali di comunicazione in modo uniforme. In particolare, le classi dedicate al protocollo MQTT sono state create per integrare facilmente questo tipo di comunicazione senza duplicare codice, favorendo modularità, riuso e la possibilità di adattare il sistema a un'architettura distribuita senza modifiche invasive.

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.