Repository github con i progetti della prima parte: https://github.com/Lorenzo-Zoccadelli/iss_lab_2025.git

Prima parte del corso e finalità

Lo scopo di questa prima fase del corso è stata quella di realizzare un'implementazione per il gioco "Conway game of life". L'idea è stata quella di partire da una semplice implementazione locale in java per poi mano a mano migliorarla, sia nella struttura del software sia per portarla su un ambiente distribuito, possibilmente basato su microservizi. In primo luogo, è stato fatto un refactoring del codice per mettere in evidenza i benefici derivanti dall'utilizzo di design pattern e tecniche di ingegneria del software come:

- Single responsability principle, in modo che ogni classe (in generale modulo) abbia un singolo compito concettuale
- Dependency inversion principle, in modo che il core businnes dipenda da delle interfacce (generalmente più stabili rispetto a modifiche) rispetto all'implementazione di moduli esterni come quelli di input/output. In generale l'obiettivo è quello di fare in modo che ogni classe dipenda da codice il più stabile possibile
- Alcuni design pattern come il pattern singleton e il pattern observer. Quest'ultimo con la possibilità di essere realizzato anche attraverso delle callback

Nello specifico il dependency inversion principle permette di realizzare il core businness in modo che in un secondo momento risulti semplice sostituire i dispositivi e i meccanismi di input/output, facilitando anche il passaggio da un ambiente locale a uno distribuito.

Infatti, una volta sistemato il codice della versione locale, isolando la parte relativa all'I/O, si è passati a una prima versione distribuita basata sul modello client-server con una gui in javascript. Questo ha portato alla necessità di introdurre un server web e un meccanismo di comunicazione tra la gui e il server applicativo che permettesse di aggiornare la griglia e inviare i comandi senza la necessità di aggiornare la pagina a ogni interazione. Per questa ragione è stato scelto di sfruttare le web socket.

Per trasformare la prima applicazione locale in quella appena descritta è stato d'aiuto l'utilizzo di un framework che permettesse di gestire in maniera più semplice ed efficace alcuni dettagli come il server web, il mapping delle websocket e la gestione delle richieste tramite dei controller specifici che si sono affiancati al controller a livello di core businness. Ciò, unito alla migliore struttura del codice ha permesso di realizzare la nuova versione in maniera piuttosto semplice. Come framework è stato scelto Spring Boot in quanto uno dei più utilizzati in java.

Successivamente si è analizzato come fosse possibile utilizzare lo stesso servizio, basato sullo scambio di dati tramite websocket, anche per implementare un client che in modo autonomo richiedesse l'esecuzione di comandi e ricevesse gli aggiornamenti dal server, aprendo la possibilità di un'interazione machine-to-machine in aggiunta a quella human-to-machine offerta dalla gui.

Infine, si è deciso modificare ulteriormente l'applicazione, cercando di implementarla basandosi su un modello ad agenti, in cui client e server sono concettualmente due semplici macchine che eseguono autonomamente e comunicano per mezzo di una terza entità broker. Quest'ultimo deve permettere a ogni agente di inviare dati a uno o più altri agenti e di riceverne in maniera indipendente. Per fare ciò è stato scelto l'utilizzo del protocollo mqtt, il quale si basa su "topic", sui quali gli agenti

possono inviare dati o registrarsi (subscribe) per essere notificati ogni qualvolta siano presenti nuovi messaggi.

In conclusione, la finalità di questa prima parte è stata quella di capire, con un approccio bottom-up, come poter passare da un'applicazione locale e concentrata a una distribuita e successivamente a un modello ad agenti, con l'ottica di portarsi sempre più verso un approccio a microservizi in cui eventualmente ogni cella della griglia possa essere gestita da un agente diverso e indipendente e che, eventualmente, possa comunicare con qualunque altro agente.

Tecnologie utilizzate e nuove competenze

Credo che la motivazione della scelta del gioco della vita di Conway come primo caso di studio per analizzare gli argomenti del corso sia legato alle seguenti ragioni:

- Si tratta di un'applicazione con il giusto grado di complessità, che permette di evidenziare molti aspetti mantenendo una dimensione e una complessità del software tutto sommato limitati
- Si presta bene ai passaggi introdotti nella prima parte, in quanto è possibile passare da un'implementazione basata su stampe su terminale a una basata su gui, fino a poter analizzare una possibile realizzazione client-server e ad agenti.
- Nello specifico la necessità dell'invio periodico di aggiornamenti ai client senza necessità che essi siano preceduti da singole richieste permette di affrontare la gestione di diversi tipi di interazione
- Si presta bene al successivo sviluppo come applicazione basata su microservizi e sull'interazione di più agenti, ottenibile ad esempio separando su più agenti la gestione delle singole celle

In questo modo è stato possibile realizzare e sperimentare le differenze tra diversi sistemi basati sugli approcci indicati: sistema locale concentrato, distribuito client-server, distribuito ad agenti autonomi e un'introduzione ai sistemi basati su microservizi

Nello specifico, a livello di tecnologie, è stato possibile approfondire e imparare (anche se in maniera non approfondita):

- Come utilizzare in modo corretto build tool come gradle (io avevo usato in passato maven, ma in questo modo sono riuscito ad approfondire anche gradle)
- Mi è stato possibile approfondire il framework Spring Boot. Avevo già utilizzato questo framework, ma è stato possibile capire meglio come utilizzarlo e quali vantaggi può portare
- Ho capito il funzionamento del protocollo mqtt che non avevo mai utilizzato prima
- Mi è stato possibile provare a utilizzare docker, anche se per ora lo so utilizzare in modo ancora un po' limitato
- A livello di modelli ho imparato e approfondito il concetto di interazione machine-to-machine, di microservizi e di sistemi basati su agenti

Nello specifico, credo che Spring Boot sia un elemento molto importante nello sviluppo software, perché permette non solo di realizzare in modo più pratico e veloce di server web e sistemi software più complessi, ma offre anche la possibilità di sfruttare la dependency injection, molto utile per esprimere le dipendenze tra i componenti a livello di configurazione senza cablarle strettamente nel

codice. Credo che Spring Boot sia un ottimo strumento e un'ottima soluzione in quanto basato su java (linguaggio comodo in quanto conosciuto da tutti gli studenti del corso).

Per quanto riguarda il linguaggio java, invece, anche se ancora largamente utilizzato, è probabilmente più obsoleto rispetto alle infinite alternative possibili. Nello specifico, la scelta del linguaggio influisce, talvolta anche in maniera pesante, sul livello di astrazione offerto al programmatore: un linguaggio con un livello di astrazione e con caratteristiche più avanzate permette al programmatore di impiegare meno sforzi nel creare tale livello di astrazione da sé. Aggiungere livelli di astrazione permette di realizzare codice più semplice, leggibile e scalabile, permettendo al programmatore di concentrarsi sul core businness della sua applicazione. Un esempio può essere lo scarso supporto di java alla programmazione funzionale, che risulta invece molto utile per realizzare il pattern observer tramite callback. Perciò, alla luce anche della scelta di Spring, una soluzione alternativa e più avanzato potrebbe essere quella di utilizzare kotlin come linguaggio, in quanto supportato sia dalla jvm sia da Spring Boot stesso.

Un altro modo per realizzare questi livelli di astrazione e nascondere la complessità dei meccanismi sottostanti può essere anche quello dell'utilizzo di librerie custom, come ad esempio **unibo.basicomm23-1.0.jar.** Questa è stata molto utile in quanto ha permesso di utilizzare il protocollo mqtt approcciandolo senza la necessità di conoscere e utilizzare le librerie di basso livello come paho. Ovviamente a livello sottostante queste librerie custom saranno a conoscenza di dettagli di basso livello, ma una volta realizzate, il loro successivo utilizzo semplificherà e renderà più veloce lo sviluppo di software a livello di businness logic. In questa prima parte del corso sono risultate particolarmente utili per nascondere i dettagli di mqtt e astrarre in protocollo specifico sottostante: a livello applicativo è necessario conoscere solo il funzionamento concettuale delle interazioni (topic, invio messaggi, subscribe...)

Ambiente distribuito e microservizi

Il sistema realizzato al termine della prima parte è sicuramente basato su un modello ad agenti distribuito, ma non credo che possa essere inteso propriamente come fondato su microservizi. Al momento infatti, tutti i servizi offerti dall'applicazione, quelli di controllo e aggiornamento della griglia, risiedono su un unico server che può interagire con molteplici client e può diventare un collo di bottiglia con difficoltà nella scalabilità. Probabilmente potrebbe essere possibile distribuire i servizi in modo tale che possano essere replicati e scalati in modo più agevole. Lo sviluppo che prevede la distribuzione della gestione di ogni cella su un host diverso potrebbe avvicinarsi maggiormente a quello che può essere definito un microservizio.

Penso quindi che l'ultima implementazione basata su agenti che interagiscono attraverso mqtt sia sulla buona strada per essere portato in un modello basato su microservizi ma che non presenti tutte le caratteristiche di tale modello.

Aspettative per la seconda parte

Credo che la seconda parte del corso si baserà maggiormente sui microservizi e come un modello basato su di essi possa essere sfruttato in maniera efficace per realizzare applicazioni e sistemi IoT basati sull'interazione con il mondo esterno di sensori e la comunicazione tra questi ultimi.

Autovalutazione: B-

In generale credo di aver capito e riportato tutti i concetti principali trattati fin'ora nella relazione, con però alcune mancanze e imprecisioni.

Non ho parlato di testing e JUnit e non ho riportato esplicitamente le differenze presenti tra la comunicazione sincrona e asincrona per quel che riguarda il progetto.

La descrizione che ho riportato sulla differenza tra linguaggi e librerie in generale mi è sembrata corretta, ma avrei dovuto indicare meglio come le librerie apportino solamente un'aggiunta di semantica creando un livello di astrazione che i linguaggi ottengono anche tramite una sintassi dedicata.

Per il resto credo di aver evidenziato i punti principali discussi a lezione.