Relazione fine fase 1 ISS 25

Andrea Grano - andrea.grano@studio.unibo.it
REPO GITHUB: | https://github.com/AndreaGrano/issLab25.git

Finalità e aspettative

In questa prima fase del corso l'obiettivo principale è stato il richiamo di concetti legati all'ingegneria del software principalmente al fine di disaccoppiare i vari componenti di un applicativo e passare da uno sviluppo "monolitico" a uno finalizzato alla realizzazione di un sistema distribuito, costituito da vari microservizi indipendentemente eseguibili e dialoganti tra loro.

È stato introdotto lo sviluppo agile, "snellendo" la documentazione e portando anch'essa verso un'evoluzione iterativa che consenta di tenere traccia dell'evoluzione di requisiti, analisi e progettazione durante lo sviluppo. Inoltre, un documento in forma più testuale e discorsiva è stato presentato come più facilmente "interpretabile" da un sistema di generazione automatica di software, aspetto che potrebbe essere alla base delle fasi successive del corso.

In ultima istanza, è stato introdotto il concetto di Machine-to-Machine Interaction (MMI), scelta che sembra sempre orientata alla ricerca di automatismi, in questo caso nel coordinamento di microservizi.

Sistemi realizzati e sperimentati

Personalmente ho sperimentato e ritoccato (introducendo le classi "Cell" e "Grid") una prima versione di GUI per il Gioco della vita di Conway, ho realizzato e sperimentato un sistema di Machine-to-Machine Interaction (MMI) per il medesimo e realizzato un microservizio di GUI indipendente interamente basata sull'interazione tramite protocollo MQTT in via di ultimazione e sperimentazione; inoltre, per quest'ultimo scenario, ho sperimentato la soluzione proposta dal professore basato sull'utilizzo "misto" di MQTT (per le interazioni tra la GUI e il microservizio del gioco) e WebSocket (per le interazioni tra il microservizio del gioco e la pagina HTML dei client).

Nuove abilità e competenze

Nell'ambito dei vari progetti le nuove competenze apprese hanno riguardato principalmente gli strumenti utilizzati, quali Gradle per la risoluzione delle dipendenze, Springboot quale framework per rendere distribuita un'applicazione locale e Docker per il deployment delle applicazioni, precedentemente noti solo nella teoria. Concettualmente, è stato più rigorosamente definita la nozione di "microservizio".

Motivazioni dietro lo sviluppo del Gioco della vita di Conway e se esso sia un sistema distribuito basato su microservizi

Il caso di studio proposto sembra opportuno. Inizialmente, il Gioco della vita di Conway ben si è prestato a una rapida implementazione monolitica (le regole, e conseguentemente il codice, sono molto semplici), per poi essere portato via via a divenire un sistema software basato su microservizi. Essendo un gioco, una prima divisione immediata in servizi distinti è stata tra la logica di business legata alle regole e l'interfaccia grafica; le regole del gioco sono evidentemente invarianti, mentre la presentazione della griglia può potenzialmente essere implementata in un'infinità di varianti, sia per aspetto che per tecnologie utilizzate. L'implementazione della GUI come microservizio ha poi da subito presentato problematiche tipiche, come la gestione dello stato; nello specifico, nella coerenza tra le griglie di clienti connessi prima dell'inizio di una partita e quelli connessi posteriormente (risolto inviando l'aggiornamento di tutte le celle per ogni epoca) e di possibili conflitti nell'impostazione dello stato iniziale se tale operazione viene resa possibile per più utenti contemporaneamente (risolto individuando, per ogni partita, un'utente "owner").

L'elemento principale del gioco è una griglia composta da un dato numero di celle e ciò potrebbe eventualmente portare a un'ulteriore evoluzione del sistema software distribuito basato su nanoservizi, i quali si occuperebbero di una singola cella. Lo scenario sarebbe interessante e probabilmente evidenzierebbe un notevole overhead nelle comunicazioni e ulteriori difficoltà nella

gestione complessiva della griglia (e.g. coordinamento dello stato delle varie celle in ogni epoca, gestione del fallimento di un nanoservizio).

Valutazioni sulla scelta di Java e framework collegati

Java e framework associati risultano a oggi ancora diffusamente utilizzati e ben collaudati. Indipendentemente dalle tecnologie utilizzate, poi, sono le nozioni presentate e i ragionamenti svolti a rivestire un ruolo chiave nella trattazione; in tal senso, l'utilizzo di un linguaggio già noto può facilitare la comprensione, relegando gli aspetti pratici a un ruolo minore.

Richiami di aspetti di ingegneria del software

Sono stati richiamati alcuni principi di design tra i SOLID quali, principalmente, Dependency Inversion Principle (e.g. l'interfaccia IOutDev, al fine di disaccoppiare l'implementazione del dispositivo di output dall'applicazione), Open/Closed Principle e, in senso lato, il Single Responsability Principle, quest'ultimo per come sono state suddivise le singole funzionalità web in vari file JavaScript.

Sono stati utilizzati il design pattern di creazione Singleton, e.g. per l'istanziazione della classe WsConwayguiLifeMqtt, al fine di avere un unico endpoint durante l'esecuzione dell'applicazione Conwayguialone, e il design pattern comportamentale Observer, usato contestualmente alla libreria custom basicomm23 per l'utilizzo dell'implementazione di Interaction per il protocollo WebSocket (le comunicazioni di tale protocollo hanno natura asincrona, pertanto l'implementazione di Interaction per WS deve essere osservabile dalla classe che la utilizza).

Le applicazioni sono state strutturate seguendo il pattern MVC (Model-View-Controller), separando le classi model contenenti lo stato dei componenti (e.g. Cell, Grid), dalle classi controller che "orchestrano" l'esecuzione (e.g. ConwayGuiControllerLifeMqtt nel progetto conwayguialone o LifeController nel progetto conway25JavaMqtt) dalle classi view che presentano l'output e acquisiscono l'input (e.g. le classi che implementano l'interfaccia IOutDev).

In generale, si è fatto, infine, riferimento all'esigenza di creare modelli del sistema, ovviamente prima di realizzarne un'implementazione, anche tramite linguaggi di metamodellazione come UML e al passaggio a uno stile di programmazione test-driven.

Ruolo delle librerie custom

Le librerie custom possono offrire un'implementazione collaudata e immediatamente accessibile di varie funzionalità; in tal senso, possono aiutare a rispettare il Single Responsability Principle e ad astrarre rispetto ai dettagli implementativi di "strumenti" utilizzati dalle classi, come, ad esempio, quelli dedicati alla gestione delle interconnessioni. Di contro, si introduce una dipendenza che può far perdere di flessibilità al codice, anche in riferimento a eventuali evoluzioni future.

Autovalutazione

Voto: B-

In generale penso di aver sufficientemente ben compreso gli argomenti presentati a lezione, aver appreso le basi dei nuovi strumenti e colto buona parte dei richiami ad argomenti noti. In tal senso, nella relazione non sono stati sottolineati alcuni aspetti (come la differenziazione tra connessioni sincrone e asincrone o JUnit) in quanto considerate nozioni già consolidate. Tuttavia, quest'ultimi sono mancanti, vi è un riferimento allo sviluppo test-driven ma non a JUnit, non era stato completamente compreso come MVC fosse un insieme di pattern e non un pattern a sé stante e, soprattutto, non si è fatto riferimento a come le librerie possano essere usate per modificare la sintassi di talune funzionalità.