Relazione ISS25

Marco Crisafulli

Nelle prime lezioni è stato introdotto il gioco Conway Life (scelto per la sua immediatezza e semplicità sia di realizzazione che di rappresentazione visiva):
Il **Gioco della Vita di Conway** è un automa cellulare creato dal matematico John Conway nel 1970. Si basa su una griglia di celle che possono essere vive o morte e si evolvono a ogni iterazione seguendo quattro semplici regole:

1. **Sopravvivenza**: una cella viva con 2 o 3 celle vive adiacenti sopravvive.
2. **Morte per isolamento**: una cella viva con meno di 2 celle vive adiacenti muore.
3. **Morte per sovrappopolazione**: una cella viva con più di 3 celle vive adiacenti muore.
4. **Nascita**: una cella morta con esattamente 3 celle vive adiacenti diventa viva.

Una prima implementazione che abbiamo visto è stata quella in javascript che aveva il fine di essere funzionale e semplice. Cito il materiale del corso:

Progetto conway25JavaScript: realizzazione in JavaScript, che permette la visualizzazione e il controllo della griglia all'interno in una pagina HTML senza l'uso di alcun server.

ConwayLife in java

Dopo aver visto questa prima soluzione al problema, abbiamo provato ad implementare lo stesso problema in java.

Abbiamo diviso il lavoro in piu' fasi:

1. Individuare i componenti per creare un Modello di gioco
2. Sviluppare e introdurre ai test per verificare che questo modello funzioni correttamente

Una volta il tutto in packages, è sorta di necessità di parlare di un dispositivo di output:
"La IOutDev interface è definita nel package della logica applicativa, come un contratto che i dispositivi di Output devono rispettare implementando in modo opportuno il metodo per visualizzare una cella." Dal Materiale del corso.
Indipendentemente dall'effettivo dispositivo di output, abbiamo creato un'interfaccia che verrà sfruttata da esso. Al momento, l'output è mostrato su terminale.

ConwayLife in java con SpringBoot

Una volta sviluppato il modello, siamo passati alla fase successiva: Introduzione a SpringBoot finalizzata alla creazione di un servizio per far connettere piu' utenti alla pagina html (non necessariamente con permessi owner) e far visualizzare il progredire del gioco anche a questi utenti terzi.

E’ sorta la necessità di un refactor del codice utilizzato fin’ora perchè il modello non ha previsto le celle come degli Oggetti ma come dei semplici numeri. Perciò sfruttando i punti di forza di Java, abbiamo trasformato in oggetti sia le celle che la griglia.
Siamo pronti ad implementare la logica spring, dopo un’attenta introduzione a SpringBootInitializer per creare lo scheletro e gradle per gestire il progetto e le dipendenze.
Cito il materiale del corso:
Il codice Java inserito come parte integrante del servizio:

- ha la stessa struttura introdotta in ConwayLife25 in Java ma con un diverso LifeController
- la classe Life non ha il compito di visualizzare le celle, compito assunto dal ConwayGuiControllerLifeLocal
- elimina ogni dispositivo di input, in quanto i comandi-utente vengono inviati tramite WebSocket
- definisce un dispositivo di output WSIoDev che implementa IOutDev interface inviando messaggi alla pagina HTML tramite WebSocket

In questo modo abbiamo un primo microservizio.

L’approccio usato è quello dell’armatura di IronMan per incassare il progetto in questa armatura cercando di modificare il meno possibile, dato che se il progetto funziona, non c’è motivo di cambiarlo radicalmente.

SpringBoot è stato scelto come framework per la sua semplicità (dato che nasconde molto codice dietro delle annotazioni) e conseguente riduzione delle righe di codice da scrivere per ottenere un servizio funzionante.

Secondo me la scelta di combinare java con SpringBoot, websocket e javascript è efficace per la sua semplicità e rapidità di sviluppo rispetto ad altre soluzioni (tipo interfacce locali JavaFX o swing), inoltre la pagina HTML è uno strumento universale indipendente dalla piattaforma hardware su cui la si visualizzi.

Librerie Custom

Per la realizzazione delle WebSocket abbiamo visto 2 approcci:
- Utilizzare le funzioni standard della libreria WebSocket scelta
- Utilizzare le funzioni di una libreria custom unibo.basicomm23

Dopo un confronto tra le 2 versioni, sono chiari i vantaggi di usare il secondo approccio:
Come primo vantaggio abbiamo la Standardizzazione dei processi, nascondiamo processi e funzioni complesse ricavate combinando le funzioni di libreria dietro delle funzioni semplici e con nomi intuitivi.
Queste funzioni di libreria custom sono perciò uguali per tutti (es. in una software house) in modo da avere uno sviluppo software coerente ed evitare possibili errori a livello “nascosto” dalla libreria, essendo la libreria un prodotto già testato e funzionante.
Personalmente mi è stato ripetuto più volte: ”non reinventare la ruota”, in questo contesto queste parole si applicano perfettamente.

Altre Tecnologie viste durante il corso

Docker è un applicazione che permette la “distribuzione” del proprio programma in maniera “semplice”.
Cito il materiale didattico: “Il deployment del servizio può avvenire mediante una immagine Docker che può essere creata ed eseguita con i seguenti comandi “.
Docker funziona creando delle immagini immutabili a partire da un DockerFile che scriviamo noi, queste immagini possono poi essere eseguite in un ambiente isolato, una “scatola”.
Abbiamo anche visto la tecnologia Docker Compose per permettere a piu’ container di comunicare su una rete “bridge”:
Questa rete bridge crea un’interfaccia di rete virtuale per il container, isolandolo dalla rete dell’host.

Interazioni M2M tramite conwayCallerWs che si connette con WebSocket al servizio e manda e riceve messaggi. Abbiamo inserito protezione chiamata ownerOff per far si che solo l’owner possa selezionare celle, avviare e stoppare il gioco, quindi evitare intrusioni.
In conwayCallerWsInteraction abbiamo utilizzato le funzioni della libreria custom unibo.basicomm23 (unibo.basicomm23.utils.ConnectionFactory) nascondendo al livello implementativo i dettagli relativi all’uso della libreria javax.websocket.

Protocollo MQTT per inviare e ricevere informazioni a/da ConwayGui come micro-servizio a sè stante.
MQTT prevede un meccanismo di Publisher/Subscriber e un brokerMqtt per connettere la gui al “mondo esterno”.
Ancora una volta abbiamo scelto di costruire astrazioni di comunicazioni di piu’ alto livello realizzandole tramite libreria Paho “nascosta” da una libreria custom.
Sperimentazione del nuovo sistema compiuta con successo tramite Docker per apprezzarne il funzionamento.

Conclusioni

Lo scopo della prima parte del corso è stato capire come implementare un programma a partire da una consegna e poi evolverlo a seconda delle necessità.
Siamo partiti da una semplice implementazione in Javascript -> Implementazione in java -> Refactor per adattamento a SpringBoot -> Introduzione Librerie Custom e WebSocket -> Docker e Mqtt.

Abbiamo sperimentato diversi protocolli di comunicazione, con i loro vantaggi e svantaggi e abbiamo provato ad implementare un’interazione M2M. Immagino queste conoscenze ci serviranno per la prossima parte del corso e in parte per il progetto finale.

Ho appreso anche dei principi utili per lo sviluppo software, per esempio l’uso di librerie custom con fine di standardizzazione e riduzione codici boilerplate e principio di IronMan per adattare un programma ad un framework.