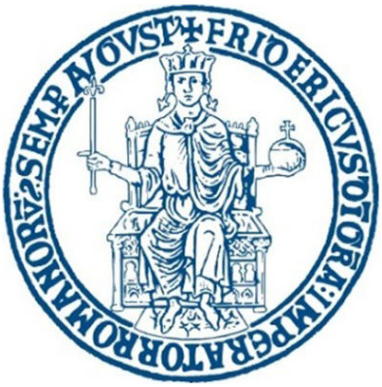
Università degli studi di Napoli Federico

Dipartimento di Ingegneria Elettrica e delle
Tecnologie dell'Informazione



Laboratorio di Sistemi Operativi: Dialogare con i robot

Giuseppe Amato [N86002167]
Luigi Ruggiero [N86003593]
Vincenzo Noviello [N86003259]

Docente: Alessandra Rossi

Anno accademico 2024–2025

Indice

1 Introduzione al progetto

Il progetto "Parlare con i robot" si occupa di sviluppare un'architettura client-server che consente all'utente di dialogare con un automa. Il sistema è composto da tre parti principali:

Il Client: è la parte del sistema che gestisce la conversazione tra l'Utente e il Robot. Il suo funzionamento si divide in due momenti principali: test di personalità e conversazione.

Il Server: Riceve le risposte dell'Utente memorizzate dal Client e calcola la personalità dell'Utente. Restituirà a sua volta parametri di personalità.

Il Robot: L'interlocutore dell'Utente. È l'interfaccia con cui l'Utente interagirà.

2 Client

2.1 Introduzione al Client

Il Client è una *skill* per il robot Furhat che gestisce la conversazione con l'utente.
Il linguaggio di programmazione supportato dalle API di Furhat è Kotlin, di conseguenza il Client è stato sviluppato in questo linguaggio.
Ogni skill è composta da *stati* in cui il robot puó transitare. A seconda dello stato in cui si trova il Robot, il suo comportamento sará diverso.
Il passaggio da uno stato all'altro è denominato *Flow*. Gli stati principali da cui è composta la skill sono:

- **Init:** Init
- **Greeting:** Greeting
- **Personality Test:** PersonalityTest
- **Conversation:** Conversation
- **Idle:** Idle

Tutti questi stati ereditano da uno stato base *Base* che gestisce le funzionalità comuni.

2.2 Flow del Client

Il robot parte nello stato *Init* e verifica la presenza di un interlocutore. In sua assenza passa allo stato Idle, altrimenti lo saluterà passando allo stato Greeting.
Nello stato Greeting, il robot chiederà all'Utente se può sottoporgli un questionario per stabilirne la personalità. In caso di risposta affermativa, passerà a PersonalityTest, altrimenti procederà direttamente con il passaggio allo stato Conversation.
In PersonalityTest, il robot sottoporrà all'Utente delle domande allo scopo di stabilire la personalità del suo interlocutore. Tali domande sono quelle del questionario TIPI, dove si chiede all'utente di rispondere su una scala da 1 a 7 quanto si identifica in determinati comportamenti o stati d'animo.
Il Client memorizza le risposte dell'Utente in un'associazione (Tratto caratteriale - Valore) ed invia il risultato del questionario a un Server. Il Server elabora le risposte calcolando un risultato che restituirà al Client.

Il Client gestirà da qui in poi la conversazione tra l'Utente e il Robot, passando allo stato *Conversation*. Qui i prompt dell'utente verranno inviati insieme a un prompt costruito in base alla personalità calcolata (se presente) a OpenAI, che restituirà la risposta da far proferire al robot.

2.3 Stati in dettaglio

2.3.1 Init

All'avvio, il Robot entra in una fase di inizializzazione:

- Imposta il numero massimo di Utenti e la distanza di ingaggio per intrattenere una conversazione.
- Imposta la lingua.
- Sceglie una voce tra le disponibili.

Dopodiché, se si utilizza un emulatore per simulare il Robot o esso individua un Utente fisico, esso entrerà nella Skill *Greeting*. Altrimenti entrerà in uno stato *"dormiente"* chiamato *Idle* in cui attenderà stimoli visivi o sonori che lo portino al *Greeting*.

2.3.2 Greeting

In questa fase, il Robot chiederà all'Utente se può sottoporgli un questionario per stabilirne la personalità.
In caso affermativo, il Robot entrerà in uno stato *Personality Test* in cui effettuerà una serie di domande.
In caso negativo, il Robot salterà il Test e passerà direttamente allo stato *Conversation*.

2.3.3 Personality Test

In questa fase il Robot sottopone all'Utente 10 domande, una per ogni tratto caratteriale, a cui l'Utente dovrà rispondere su una scala da 1 a 7.
Il test viene costruito dapprima associando ai 10 tratti di personalità una stringa omonima. Il Robot quindi pronuncerà la domanda riguardante quel tratto grazie a questa associazione.
La risposta dell'Utente viene registrata in un'ulteriore associazione: un valore intero per ogni tratto di personalità.
Terminate le domande, il Robot ringrazierà.
A questo punto viene stabilita una connessione al server che si occuperà di elaborare i dati ottenuti. Il protocollo utilizzato è TCP.
Il client converte l'associazione tratto-valore in un JSON testuale

attraverso un'apposita libreria. Viene quindi convertito in array di byte. Questo perché i socket TCP trasmettono dati in formato binario. L'array di byte viene inviato al server tramite il flusso di output del socket.
La risposta attesa dal server viene salvata in una stringa mutable e passata allo stato *Conversation*.

2.3.4 Conversation

In questa fase il client recupera una key salvata in un file locale. Ciò avviene tramite una variabile d'ambiente salvata sul sistema.
Lo scopo è quello di configurare un'istanza del client OpenAI utilizzando una chiave API. Tale client viene utilizzato per interagire con l'API di OpenAI, vero interlocutore dell'utente.
Nel caso in cui l'utente salti il questionario, la conversazione inizia senza alcun input preliminare per ChatGPT. Altrimenti, viene richiesto di utilizzare le emoji e di rispondere in modo conciso, includendo anche il tratto di personalità emerso dal test.
Le emoji sono state raggruppate in insiemi, ciascuno dei quali associato a una specifica gesture che il robot può eseguire. Quando ChatGPT risponderà all'utente utilizzando delle emoji, queste verranno sostituite dalla gesture corrispondente, che il robot eseguirà durante la pronuncia della frase.
Alla termine della conversazione, quando l'utente si allontana o pronuncia *"Arrivederci"*, il robot ritorna nel suo stato dormiente *Idle*.

3 Server

3.1 Introduzione al Server

Il server rappresenta l'unità di elaborazione del sistema sviluppato in cui convergono le risposte raccolte tramite il questionario della personalità (clicca qui per tornare alla sezione dei dettagli del calcolo della personalità) e dove avviene poi il successivo calcolo della suddetta. Infine, il server invia i dati calcolati nuovamente all'unità Client del nostro sistema. Il linguaggio scelto per la scrittura del Server è stato il linguaggio C.

3.2 Avvio del Server

Viene Dockerizzato e avviato (clicca qui)

3.3 Il main

Il main esegue la funzione "runServer()" presente nel file server.c.

3.4 *Core* del Server

Il server (la sua strutturazione così come i suoi protocolli di comunicazione) viene sviluppato all'interno dei file *server.c* e *server.h*. Da notare nel file *server.c* (oltre ai già citati protocolli di comunicazione) è la funzione denotata dalla firma *"serveRequest"* che recupera dal client un buffer in cui raccoglie il file Json contenente le risposte date dall'utente nelle domande che gli sono state sottoposte per il questionario della personalità.

3.4.1 Protocollo di connessione del server

Il server utilizza un protocollo di connessione TCP (Transmission Control Protocol), uno degli standard per le comunicazioni, per la trasmissione dei dati. Protocollo che ci permette di avere anche una grande affidabilità sui dati trasmessi. Sappiamo infatti che se un pacchetto si perde o arriva corrotto, TCP lo rileva e lo ritrasmette. Importante feature del protocollo TCP è quella che riguarda l'interazione tra le parti. Fin tanto che la socket non viene chiusa, la connessione rimane attiva. Volevamo lasciare attiva la connessione mentre il server elabora i dati della personalitá, e questo protocollo ci permette di raggiungere esattamete questo obiettivo. E' stato impostato un limite per le connessioni in ingresso, ovvero: 10.

3.4.2 I segnali

Per avere una corretta e sicura terminazione dell'esecuzione, abbiamo definito degli handler per alcuni segnali. *Sigint* per le interruzioni manuali date dall'utente (ctrl+c), che termina con successo il programma, *Sigusr1* per gestire gli errori del processo padre, *Sigusr2* per gestire gli errori dei processi figli. I primi due chiudono la socket del padre mentre il terzo solo quella del figlio che ha generato errore.

3.5 I processi

Altra scelta implementativa da sottolineare è sicuramente quella legata ai processi. Si è scelto di utilizzare questi ultimi e non i thread per più ragioni:

- *Isolamento della memoria:* ogni processo ha un'area di memoria dedicata, e ciò permette che un errore non influenzi gli altri processi;
- *Stabilità:* se un processo va in crash, non influisce su tutti gli altri;
- *Comunicazione controllata:* l'utilizzo di processi riduce il rischio di condizioni di gara e problemi di sincronizzazione.

3.6 Il "calcolo" della personalità

Ultimo punto da affrontare è l'elaborazione, da parte del server, della personalità dell'individuo con cui il client interagisce. Sommariamente, il server riceve direttive su come manipolare i dati che vengono ricevuti dopo l'interazione con l'intelligenza furhat. Una volta ricevuto il JSon che contiene le risposte, si occupa di estrapolare queste ultime e di associare il valore numerico corrispondente per ogni tratto di personalità considerato. Fatto ciò, esegue il calcolo secondo le direttive del test di personalità scelto per questo progetto, ovvero il TIPI. Infine, invia il tratto calcolato al client così da assumere una personalità che meglio permetta la riuscita di una buona interazione con l'individuo che desidera usare l'applicativo.

4 Implementazione

4.1 Dockerfile Server

Il Dockerfile del Server si compone di due parti: creazione ed esecuzione di un'applicazione Server attraverso due appositi containers.

4.1.1 Creazione

Per la creazione si è scelto di utilizzare come immagine base *alpine:3.21*, una distro Linux molto leggera ed efficiente.
Una volta copiato il codice sorgente nel container, per la compilazione è stato installato un package chiamato build-base contenente tutti gli strumenti per la compilazione C/C++. La compilazione viene velocizzata utilizzando tutti i core disponibili (*make -j*).

4.1.2 Makefile

Nel makefile vengono abilitati warning e ottimizzato il codice.
Degno di nota l'utilizzo del flag *sanitize* per evitare il crearsi di memory leaks.

4.1.3 Esecuzione

A questo punto viene creata un'immagine pulita (sempre con alpine:3.21) con il solo file eseguibile. Viene quindi esposto il port 9999 del server.
Sono stati creati un gruppo e un utente *nonroot* per questioni di sicurezza. Attraverso di essi viene eseguito il /main.

4.2 Docker Client

Per il Client è stata utilizzata una strategia molto simile a quella per il Server: l'esecuzione infatti avviene in un'immagine dotata del solo Java Runtime Environment. Mentre per la compilazione viene utilizzata una versione che contiene solo la JDK 11.

4.2.1 Compilazione

È stata utilizzata un'immagine leggera: eclipse-temurin:11, JDK completo.
Il Client viene compilato con *Gradle Wrapper* o *Gradlew*, uno script che permette di effettuare build in locale senza il bisogno di installare Gradle, anche per questioni di portabilità.
In Gradlew è stato importato un plugin shadowJar che crea un JAR eseguibile autonomo che viene poi passato all'immagine successiva.

Nota: il JAR essendo in realtà una skill di Furhat ha estensione *.skill*.

4.2.2 Esecuzione

Per l'esecuzione è stata utilizzata un'immagine basata su eclipse-temurin:11-jre che contiene solo l'ambiente di esecuzione. Ricevuto quindi il JAR eseguibile, prima di passare all'esecuzione vengono ricevute due variabili d'ambiente: ROBOT_ADDRESS e SERVER_ADDRESS, indicanti gli indirizzi IP degli omonimi componenti.
Nota: L'indirizzo del robot viene passato alla JVM, mentre l'indirizzo del server viene passato al main.

4.3 Docker Compose

Vengono definiti due servizi principali: Client e Server.
Al server viene inoltrato il port 9999 mentre nel Client è stata aggiunta l'opzione dipendente dal Server affinché venisse avviato per primo quest'ultimo.
Il container condivide la rete del sistema host per permettere la comunicazione con il robot di Furhat.
Viene passato un segreto, un file contenente la key per l'API di OpenAI oltre alle due variabili d'ambiente con gli indirizzi IP del Server e del Robot.