1. Introduzione e formulazione del problema

Il problema presentato consiste nella progettazione e realizzazione di un applicativo che implementi una chat punto-punto. Il quesito si propone su più livelli di complessità: si parte dalla definizione di un software basilare che simuli un servizio di messaggistica istantanea molto povero di funzionalità, fino a raggiungere, nella sua versione più avanzata, una chat multicast con l'inserimento di opzioni e operazioni applicabili sulla conversazione.
Analizzando le richieste, si è scelto di approcciarsi al problema tramite una metodologia Scrum, partendo dalla progettazione ed implementazione della versione basilare, per poi passare,nel caso di successo, ad una versione più avanzata.

Le funzionalità che vengono richieste consistono nella creazione di una chat bidirezionale, tra due host, con la possibilità di:

- impostare un nome utente/autore, visibile all'altro interlocutore;
- poter inviare uno smile o un like come messaggio;
- trasmettere l'ultimo messaggio ricevuto;
- impostare e gestire uno stato di disponibilità nella ricezione dei messaggi;
- poter chiudere tramite messaggio la connessione;

Queste richieste, analizzandole, raccolgono un insieme logico di descrittori e funzioni che vanno a caratterizzare le due entità.

Il programma si basa sulla programmazione di rete e come tale, necessita la presenza di due entità particolari: il client e il server. Queste due classi si assomigliano per molti aspetti, tra cui la lista degli attributi da cui sono composti. L'unica differenza, che complica la questione, è il come viene valorizzato l'oggetto socket all'interno di queste entità. Nel caso del server, verrà ottenuto dal metodo accept(), mentre per quanto riguarda il client, l'istanza verrà creata. Il problema lo si ritrova nel momento in cui si intende attuare un processo di generalizzazione sulle due entità sopra citate, rendendo ereditati anche i metodi per la lettura e scrittura di un messaggio. Queste funzioni non presentano differenze nel codice e quindi è necessario se possibile renderli come metodi della superclasse. Questo però comporta una serie di problematiche che risiedono nella valorizzazione degli attributi in sé, che renderebbero la gerarchia e la logica della connessione errata ed instabile. Per ovviare a questa problema si è pensato di attuare un'inversione di logica. In poche parole, si è deciso creare una classe gestore, che raccogliesse tutti i metodi che gestiscono la scrittura, lettura ed interpretazione dei messaggi, inclusi i possibili scrivi() e leggi(), e quindi utilizzata da entrambi le entità come attributo. Il problema quindi si risolverebbe inserendo nella lista dei parametri formali di tali metodi l'host stesso. Con questa soluzione si risolverebbero tutti i problemi emersi in questa fase di progettazione riguardante la gerarchia delle classi.
Di conseguenza, dopo questo ragionamento sulla struttura logica del programma, il risultato finale è un package composto da due entità distinte: Client e Server. Quest'ultime sono l'estensione della classe astratta Host. L'oggetto che implementa tutte le funzioni tipiche della chat è il Gestore.
Un'altra scelta progettuale che si è dovuti affrontare è il come gli host verranno istanziati e avviati. Le due opzioni possibili sono: la concezione delle due entità come classi avviabili individualmente tramite metodo statico o la definizione sempre di queste come thread. Entrambe le soluzioni sono efficaci, ma la seconda scelta rimane più complessa da implementare, perché l'uso di processi concorrenti comporta la scesa in campo di una serie di strumenti e metodologie di approccio che andrebbero a complicare e rendere più

difficoltosa la progettazione. Per questo, analizzando anche le richieste, risulta eccessivo l'uso di una programmazione concorrente, essendo la chat punto-punto.

2. Descrizione dell'architettura dell'applicazione (componenti hardware e software)

2.1 Attori

Come si può osservare, analizzando anche superficialmente il progetto, sono presenti due attori all'interno di questo applicativo: il Client e il Server.

Server

Il Server è colui che all'interno dell'architettura Client/Server permette l'apertura della connessione, rimanendo in ascolto su una porta prefissata, in attesa di richieste da parte di un client. All'interno del sistema, non usa altro che gli attributi su cui è basato, necessari per l'esecuzione delle funzioni per cui è stato progettato. In particolare, gli attributi fondamentali sono gli stream di input e output che permettono lo scambio e l'interazione con l'altra entità/interlocutore di informazioni.

L'entità, per come è stata progettata, è direttamente avviabile e risulta quindi indipendente dall'altro attore, client. All'avvio di questo, ci sarà una fase di attesa di un cliente che richieda una connessione e solo successivamente sarà possibile per l'attore in questione mandare e ricevere messaggi. Queste operazioni saranno svolte periodicamente finché la connessione non terminerà per volere di uno degli host o per errore.

Tutte le informazioni necessarie per l'attore per il suo corretto funzionamento vengono ottenute dall'utilizzo degli attributi al suo interno (flussi di input e output e socket).

Client

Il Client è colui che all'interno dell'architettura Client/Server richiede la connessione ad un server su una determinata porta, che deve essere uguale in entrambi i lati. Questo attore basa il suo funzionamento e la sua operatività sugli attributi al suo interno. Simile al server per composizione e funzionamento, gli attributi che maggiormente ricoprono un ruolo rilevante sono i flussi di input ed output che permettono di comunicare ed interagire con l'altro attore all'interno del sistema.

L'entità in questione è stata progettata per essere avviata in modo autonomo dal server e il resto del package, risulta però necessaria la presenza dell'altra entità in esecuzione e in attesa di una richiesta. Questo vincolo è dettato dall'architettura di rete TCP/IP che rende subordinato il client al server.

Con l'instaurazione della connessione sarà possibile inviare e leggere i messaggi inviati dall'altra entità. Queste attività possono essere considerate periodiche, perché cicliche. Il programma sarà in esecuzione finché uno dei due host deciderà di chiudere la connessione e quindi interrompere la chat.

Tutte le informazioni necessarie per l'attore, per il suo corretto funzionamento vengono ottenute dall'utilizzo degli attributi al suo interno (flussi di input e output e socket).

2.2 Casi d'Uso

In questo applicativo sono presenti principalmente cinque casi d'uso: connetti, attendi, leggi, scrivi e chiudi.

Caso d'Uso: Apertura della connessione e attesa
ID: UC1
Attori: Server
Descrizione: l'attore Server instaura una connessione secondo le regole del protocollo TCP e attende su una determinata porta possibili richieste.
Pre-condizioni: le premesse che devono essere soddisfatte per la corretta esecuzione di tale caso d'uso consistono nell'istanziamento di un oggetto Server e l'invocazione del metodo che contiene le istruzioni necessarie.
Azioni: l'attore tramite l'utilizzo di ServerSocket e dei metodi ad esso associati, attende che un attore Client richieda una connessione su una porta concordata precedentemente. Nel momento in cui avviene tale azione, il protagonista accetta e instaura una connessione stabile.
Post-condizioni: nel momento in cui questa operazione avviene con successo, si otterrà un oggetto Socket che rappresenta in modo astratto la connessione stabilita tra i due attori e di conseguenza la valorizzazione di alcuni attributi dell'entità, come per esempio i flussi di input e output. Nel caso di un insuccesso dell'operazione verrà restituito solamente un messaggio di errore con relative informazioni.

Caso d'Uso: Richiesta di connessione
ID: UC2
Attori: Client
Descrizione: l'attore Client richiede la connessione ad un Server su un determinato IP e una precisa porta, secondo le regole del protocollo TCP.
Pre-condizioni: le premesse che devono essere soddisfatte per la corretta esecuzione di tal caso d'uso consistono nella precedente creazione di un oggetto Client e dell'esistenza di un processo Server già avviato che accogli la richiesta dell'attore.
Azioni: l'attore tramite l'utilizzo dell'oggetto Socket e dei metodi ad esso associati, richiede l'istaurazione di una connessione su una determinata porta e su un preciso IP. Nel momento in cui avviene tale azione, il protagonista termina l'esecuzione di tale operazione creando un'istanza di GestoreChat associato a tale attore.
Post-condizioni: nel momento in cui questa operazione avviene con successo, si otterrà un oggetto Socket che rappresenta in modo astratto la connessione stabilita tra i due attori e la conseguente valorizzazione di alcuni attributi dell'entità come per esempio i flussi di input e output. Nel caso di un insuccesso dell'operazione verrà restituito solamente un messaggio di errore con relative informazioni.

Caso d'Uso: Scrivi messaggio
ID: UC3
Attori: Server/Client
Descrizione: L'attore invia un messaggio all'altra entità con cui ha instaurato una connessione.
Precondizioni: Per far sì che questa operazione sia eseguibile è necessario che entrambe gli attori siano avviati e la connessione instaurata. Di conseguenza deve essere già avvenuta la valorizzazione di tutti gli attributi utili allo scambio, come i flussi di I/O
Azioni: l'attore protagonista di questo caso d'uso, andrà ad inviare un messaggio all'altra entità collegata. L'invio verrà effettuato tramite l'utilizzo dell'oggetto e del flusso di output associato alla connessione. Esistono due tipologie di messaggi: normali ed interpretati. La differenza sostanziale sta nel fatto che alcune funzionalità vengono richiamate da una determinata stringa, che dovrà essere analizzata. Se questa non corrisponderà a nessun comando particolare, verrà inoltrata così com'è. Nel caso contrario, a seconda della stringa scritta saranno eseguite le operazioni associate. Ecco di seguito le azioni richiamabili tramite messaggio: • modifica del nome dell'autore del messaggio; • invio di uno smile; • invio dell'ultimo messaggio ricevuto; • modifica dello stato di disponibilità nella ricezione dei messaggi; • ristampa del menu; • chiusura della chat;
Postcondizioni: Il risultato di questa operazione di invio, muterà a seconda del messaggio letto da tastiera. Di seguito elencherò le varie postcondizioni: • effettiva modifica del nome dell'entità, che verrà comunicata anche all'interlocutore; • ricezione da parte dell'interlocutore di uno smile; • ricezione da parte dell'interlocutore del messaggio inviato precedentemente; • blocco di tutti i messaggi in entrata. L'entità fino al cambio di tale stato non riceverà nessun messaggio dall'interlocutore; • ristampa sul terminale il menu; • il flusso di messaggi viene interrotto e avvisato tramite stampa su terminale. Alcune delle postcondizioni citate (2, 3 e la 6) dipendono in modo stretto alla corretta esecuzione del caso d'uso UC4

Caso d'Uso: Leggi messaggio
ID: UC4
Attori: Client/Server
Descrizione: L'attore ricevente un messaggio sul flusso di input, legge tale stringa e la rende visibile sul terminale, stampandola.
Precondizioni: Per far sì che questa operazione sia eseguibile è necessario che entrambe gli attori siano avviati e la connessione instaurata. Di conseguenza deve essere già avvenuta la valorizzazione di tutti gli attributi utili allo scambio, come i flussi di I/O. E' fondamentale per l'esecuzione del caso d'uso l'invio di un messaggio da parte dell'altra entità partecipante alla chat.
Azioni: utilizzando lo stream di input associato al Socket, verrà letto tramite l'uso di classi apposite il messaggio inviato e di seguito stampato a schermo.
Postcondizioni: Il caso d'uso sarà correttamente eseguito se il messaggio inviato verrà stampato sul terminale del destinatario (l'attore protagonista del caso d'uso).

Caso d’Uso: Chiudi connessione
ID: UC5
Attori: Client e Server
Descrizione: Uno dei attori, inviando la particolare stringa per la chiusura della connessione, permetterà di terminare lo scambio di messaggi da ambo le parti.
Precondizioni: Per far sì che questa operazione sia eseguibile è necessario che entrambe gli attori siano avviati e la connessione instaurata. Di conseguenza deve essere già avvenuta la valorizzazione di tutti gli attributi utili allo scambio, come i flussi di I/O.
Azioni: L’attore servendosi del caso d’uso UC3, può inviare una particolare stringa che permetta di concludere la connessione. Se tale comando viene immesso da uno delle due entità coinvolte all’interno del sistema, verrà comunicato tramite invio di un particolare messaggio all’interlocutore. Di seguito entrambe gli attori tramite l’uso di particolari primitive adibite per tale scopo chiuderanno la connessione affidabile da entrambi i lati, seguendo le regole del protocollo TCP.
Postcondizioni: nel caso di una chiusura della connessione, sarà impossibile inviare ulteriori messaggi all’interlocutore. La corretta esecuzione del caso d’uso darà come risultato una stampa su terminale di conferma.