Istituto Tecnico Tecnologico Statale "Alessandro Volta"
Scuol@2.0
Chimica, Materiali e Biotecnologie
Grafica e Comunicazione
Trasporti e Logistica
Via Assisana, 40/E - loc. Piscille - 06135 Perugia
Centralino 075.31045 fax 075.31046 C.F. 80005450541
www.avolta.pg.it
voltauffici@tin.it dirigente@avolta.pg.it
Meccanica, Meccatronica ed Energia
Elettronica ed Elettrotecnica
Informatica e Telecomunicazioni

Corso di

Tecnologie e Progettazione di Sistemi Informatici e di Telecomunicazioni

Classe 5AINF

A.S. 2016/2017

Relazione Progetto

"TCPChat"

Studenti: Matteo Calosci

e Diego De Leonardis

Docente: Monica Ciuchetti

Documento di definizione del problema (SRS)

1. Introduzione

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.

2. Formulazione del problema

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.

3. 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 apre la 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 i flussi di input e output che permettono lo scambio di informazioni e l'interazione con l'altro interlocutore.

L'entità, per come è stata progettata, è direttamente avviabile e risulta quindi indipendente dall'altro attore. All'avvio di questo, ci sarà una fase di attesa. Solo nel momento in cui un cliente richieda una connessione e questa venga accettata sarà possibile per l'attore in questione mandare e ricevere messaggi. Queste operazioni saranno svolte periodicamente finché la comunicazione

non terminerà per volere di uno degli host o per errore.

Tutte le informazioni necessarie per l'attore e 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 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, risulta però necessaria la presenza dell'altro attore in esecuzione e in attesa di una richiesta per l'instaurazione di una connessione. Questo vincolo è dettato dalle regole del protocollo TCP che rende il client subordinato al server.

Con l'instaurazione della connessione sarà possibile inviare e leggere i messaggi scambiati con l'altra entità. Queste attività sono periodiche. 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 e per il suo corretto funzionamento vengono ottenute dall'utilizzo degli attributi al suo interno (flussi di input e output e socket).

4. Scenari e casi d'uso

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. Di conseguenza si otterrà anche 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'instaurazione di una connessione su una determinata porta e su un preciso IP. Nel momento in cui avviene tale azione, il protagonista termina l'esecuzione dell'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.
Pre-condizioni: per far sì che questa operazione sia eseguibile è necessario che entrambi 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 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: a. modifica del nome dell'autore del messaggio; b. invio di uno smile; c. invio dell'ultimo messaggio ricevuto; d. modifica dello stato di disponibilità nella ricezione dei messaggi; e. ristampa del menu; f. chiusura della chat;
Post-condizioni: il risultato di questa operazione di invio, muterà a seconda del messaggio letto da tastiera. Di seguito elencherò le varie post-condizioni: a. effettiva modifica del nome dell'entità, che verrà comunicata anche all'interlocutore; b. ricezione da parte dell'interlocutore di uno smile;

c. ricezione da parte dell'interlocutore del messaggio inviato precedentemente;
d. blocco di tutti i messaggi in entrata. L'entità fino al cambio di tale stato non riceverà nessun messaggio dall'interlocutore;
e. ristampa sul terminale del menu;
f. il flusso di messaggi viene interrotto e avvisato tramite stampa su terminale.

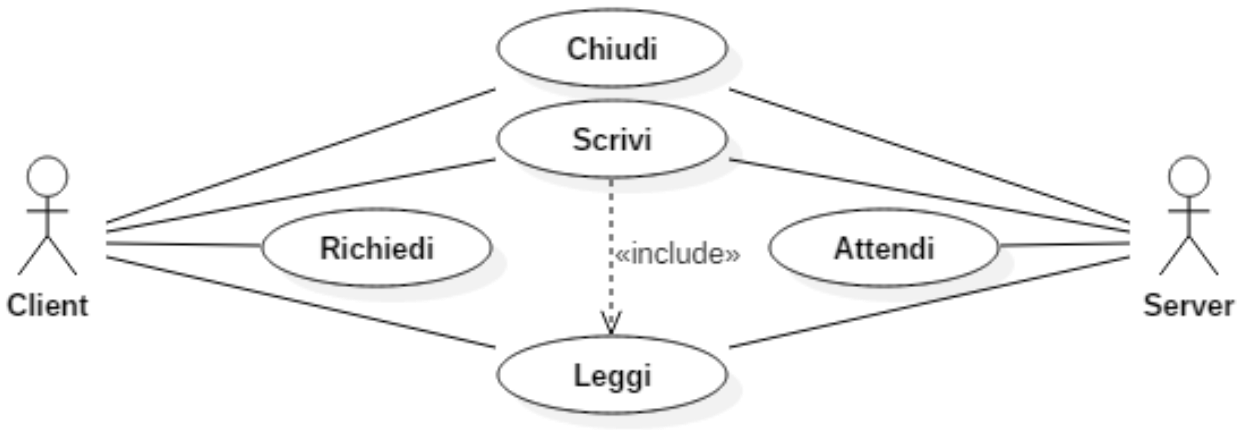
Le postcondizioni citate dipendono strettamente alla corretta esecuzione del caso d'uso UC4.

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.
Pre-condizioni: 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 metodi appositi il messaggio inviato e di seguito stampato a schermo. Esistono però due particolari casi da analizzare: ▪ nel caso in cui l'entità in questione non è disponibile il messaggio non viene stampato; ▪ nel caso in cui il messaggio corrisponda alla particolare stringa che richiama la chiusura della connessione vengono eseguite le operazioni dovute;
Post-condizioni: il caso d'uso sarà correttamente eseguito se il messaggio inviato (escludendo i casi affrontati nelle "azioni") verrà stampato sul terminale del destinatario.

Caso d'Uso: Chiudi connessione
ID: UC5
Attori: Client e Server
Descrizione: uno dei due attori, inviando la particolare stringa per la chiusura della connessione, permetterà di terminare lo scambio di messaggi da ambo le parti.
Pre-condizioni: 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 permette di concludere la connessione. Se tale comando viene richiamato da uno delle due entità coinvolte all'interno del sistema, verrà comunicato all'interlocutore. Di seguito entrambi gli attori, tramite l'uso di particolari primitive adibite per lo scopo, chiuderanno la connessione affidabile da entrambi i lati, seguendo le regole del protocollo TCP.
Post-condizioni: 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.

5. Diagramma dei casi d'uso



6. Vincoli

Si parte con il presupposto che l'applicativo descritto in questo documento delle specifiche è aggiornabile con nuove funzionalità. Il progetto proposto in sé è suddiviso in più livelli di complessità. Si ipotizza la possibilità di aggiungere altre azioni e comandi all'interno della chat, rendendola più ricca. Questa espansione non risulta essere ostica né difficoltosa per via della presenza di commenti in stile Javadoc e di una documentazione dettagliata.

Una possibile modifica che non è stata ancora implementata, ma che era stata progettata, è l'aggiunta dell'invio di allegati.

Si evidenzia come l'applicativo sia stato progettato per una connessione punto-punto. Nel caso di modifica della tipologia di chat, con passaggio da una comunicazione uni-cast a multi-cast, sarà necessario rivoluzionare tutto il codice e la gerarchia delle classi del progetto. Si può affermare come la definizione dell'applicativo dal punto di vista delle entità sia un vincolo che lega questo alla tipologia della chat.

7. Prototipo dell'interfaccia

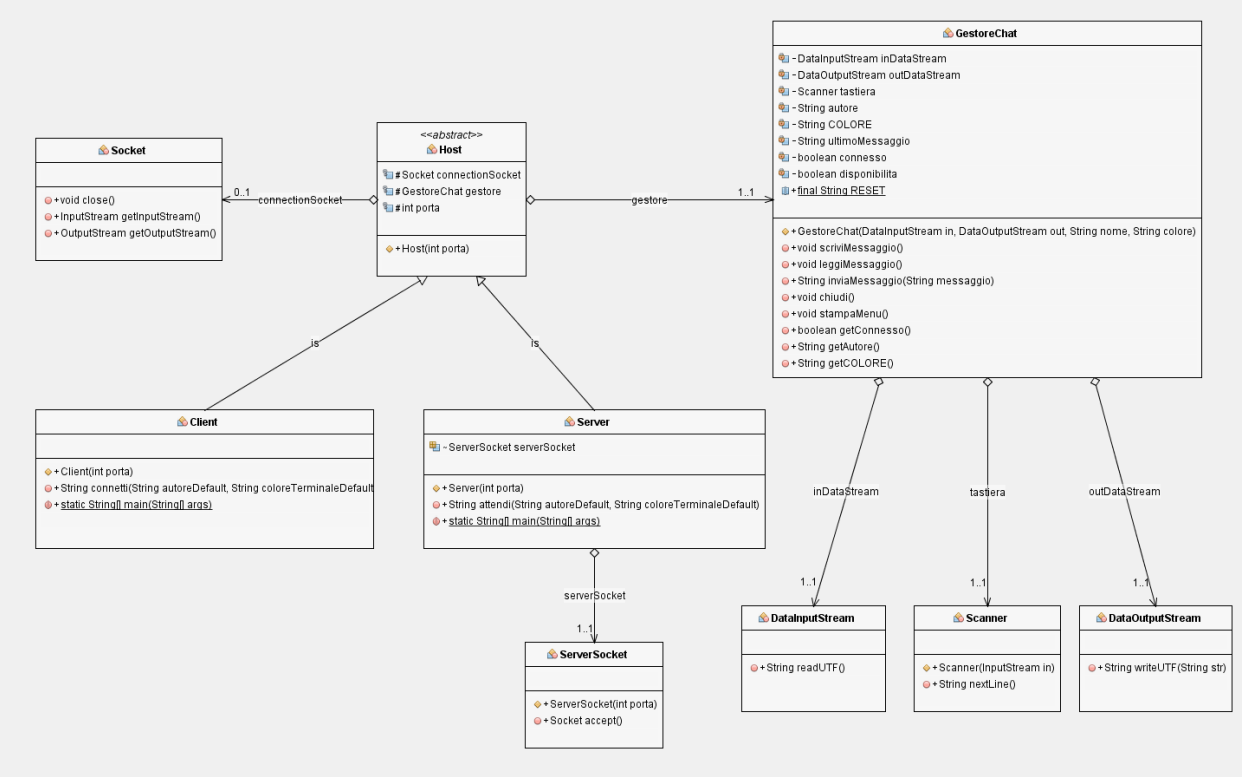
Non essendo richiesta nessuna interfaccia grafica, da parte delle specifiche, si è deciso di implementare quella da linea di comando. Si premette che questa scelta progettuale, anche se semplifica di molto il codice, limita di gran lunga il fattore estetico dell'applicativo.
Da richiesta del documento datoci, è stato deciso di definire una stampa simile a quella di una normale chat testuale, colorando con due tonalità diverse i messaggi delle due entità partecipanti alla comunicazione. In questo modo sarà possibile distinguere chi scrive cosa.

Documento di progettazione ed implementazione dell'applicazione:

1. Descrizione dei componenti hardware e software

Per lo sviluppo di questo applicativo è stato utilizzato un linguaggio di programmazione Java con una piattaforma JDK 1.8. È stato impiegato come IDE NetBeans 8.1. L'ambiente su cui si è programmato è Windows.

2. Diagramma delle classi



3. Commento del codice

I commenti del codice sono disponibili direttamente nel Javadoc. Clicca qui per visualizzarli.

4. Test dell'applicazione

Durante lo sviluppo di questo applicativo, sono stati affrontati vari problemi in fase di progettazione che hanno richiesto tempi e risorse non indifferenti.

La maggior parte degli incontri tra i vari membri del team dedicato allo sviluppo di tale progetto, è stata dedicata all'analisi e progettazione di una struttura efficiente e giusta, cercando di sfruttare al massimo il concetto di

ereditarietà insito nel paradigma ad oggetti.

L'applicativo si basa sulla programmazione di rete e come tale, necessita la presenza di due entità particolari: un processo client e un processo server. Queste due classi si assomigliano per molti aspetti, in particolare per 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 dall'accettazione della richiesta di connessione da parte del client, mentre per quanto riguarda quest'ultimo, l'istanza verrà creata. Il problema lo si ritrova nel momento in cui si intende attuare un processo di generalizzazione delle 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 definirli come metodi della classe-madre. Questo però comporta una serie di problematiche che risiedono nella valorizzazione degli attributi in sé. Le varie possibili soluzioni trovate avrebbero reso 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 GestoreChat, che raccogliesse tutti i metodi che gestiscono la scrittura, lettura ed interpretazione dei messaggi. Il problema quindi si risolve associando ad ognuno dei due host un'istanza di tale classe come attributo. 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 è stata necessaria affrontare è il come gli host verranno istanziati e avviati. Le due opzioni possibili erano: la concezione delle due entità come classi avviabili individualmente tramite metodo statico o la definizione di queste come processi leggeri. 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.