Università degli Studi di Catania

Facolta di Ingegneria Informatica

A.A.2004/2005

Progetto di Ingegneria del Software: Talk-Talk

Coco Ignazio Andrea – 616/000075

Professore: A.Calvagna

Definizione del problema e specifica dei requisiti

Il problema verrà definito e fondato sulla base dei **requisiti informali specificati dal docente** che per completezza vengono riportati di seguito.
Lavorando su queste direttive, formalizzando e completando si ricaverà il domino applicativo e quindi le specifiche dei requisiti del progetto.

Si realizzi una applicazione Java client/server, usando Sockets, RMI o CORBA, che consenta a due o più utenti remoti di comunicare simultaneamente attraverso lo scambio di messaggi digitati sulla tastiera, una riga di testo per volta.

Si assuma che tutti gli utenti siano univocamente identificati da un nome (una stringa) come ad esempio "andrea". Per potere ricevere i messaggi a lui destinati, l'utente andrea digita sulla console il comando:

> listen andrea (invio)

con l'effetto che di seguito verranno stampati a video tutti i messaggi ricevuti, nell'ordine in cui li riceve, nel seguente formato: <nome mittente>:<messaggio>

Se andrea vuole iniziare la comunicazione verso un altro utente, del quale conosce già per ipotesi il nome (ad es., Giorgio), semplicemente aprirà una

nuova console e digiterà il comando:

> talk Giorgio (invio)

seguito dal testo che vuole inviargli, linea dopo linea. Per indicare la volontà di terminare l'invio di messaggi digiterà invio non preceduto da alcun carattere, per indicare una linea vuota.

Scomposizione del problema

Obiettivo del programma

Si realizzi una applicazione Java client/server, usando Sockets, RMI o CORBA, che consenta a due o più utenti remoti di comunicare simultaneamente attraverso lo scambio di messaggi digitati sulla tastiera, una riga di testo per volta.

Assunzioni iniziali

Si assuma che tutti gli utenti siano univocamente identificati da un nome (una stringa) come ad esempio "andrea".

Requisiti informali

Per potere ricevere i messaggi a lui destinati, l'utente andrea digita sulla console il comando:

1 °

> listen andrea (invio)

con l'effetto che di seguito verranno stampati a video tutti i messaggi ricevuti, nell'ordine in cui li riceve, nel seguente formato: <nome mittente>:<messaggio>

2°
Se andrea vuole iniziare la comunicazione verso un altro utente, del quale conosce gia' per ipotesi il nome (ad es., giorgio), semplicemente aprira' una nuova console e digitera' il comando:

> talk giorgio (invio)

seguito dal testo che vuole inviargli, linea dopo linea. Per indicare la volonta' di terminare l'invio di messaggi digitera' invio non preceduto da alcun carattere, per indicare una linea vuota.

Suggerimenti:

Scrivere il programma sotto forma di due componenti:

uno server (listen) che riceve messaggi e li stampa su una finestra di console e

l'altro client (talk) che li invia.

Del talk potrei avere in esecuzione piu' istanze contemporaneamente, una per ogni utente con cui voglio parlare

(provarlo almeno con tre utenti contemporanei).

Scelte progettuali iniziali

Per il programma in questione si è scelto di usare le socket in quanto metodologia di scambio messaggi tra client e server semplice e facile da apprendere. Sono state escluse le ulteriori scelte RMI/CORBA in quanto introducono una maggiore complessità e dispongono di funzionalità avanzate che risultano ridondanti in questo sistema.

Ogni utente disporrà di un programma server aperto tramite una console in cui si leggeranno tutti i messaggi diretti all'utente.

Inoltre si potranno avere in altre finestre di console più istanze del lato client dell'applicazione che si occupa di instaurare connessione verso utenti online di cui si conosce il nome. Tramite questo programma l'utente potrà inviare i messaggi.

L'assunzione che ogni utente sia identificato da una stringa che lo identifica deve essere completata, in quanto dovremo gestire il caso in cui due o più utenti tentano di accedere contemporaneamente con lo stesso identificativo che chiameremo nickname.

Requisiti Formali

A questo punto vanno formalizzate le due specifiche fondamentali del programma, che rappresentano la base di funzionamento che il committente richiede. La prima richiesta

> listen andrea (invio) :: con l'effetto che di seguito verranno stampati a video tutti i messaggi ricevuti, nell'ordine in cui li riceve, nel seguente formato: <nome mittente>:<messaggio>

rappresenta una sorta di login verso una macchina server che mantiene l'elenco degli utenti connessi. Con questo comando (listen nome_utente) l'utente comunica al server la sua presenza nella chat, ovvero il suo stato di on-line. Sulla macchina server che mantiene l'elenco degli utenti verrà associato ad ogni utente collegato il suo indirizzo IP e la porta su riceverà i messaggi. Si è inoltre inserito un metodo per chiudere il programma in modo corretto, ovvero si è scelto di premere il tasto e per la chiusura del Listen avviando una procedura che chiuda tutte le connessioni.

Se andrea vuole iniziare la comunicazione verso un altro utente, del quale conosce gia' per ipotesi il nome (ad es., giorgio), semplicemente aprira' una nuova console e digitera' il comando:

> talk giorgio (invio)

seguito dal testo che vuole inviargli, linea dopo linea. Per indicare la volonta' di terminare l'invio di messaggi digitera' invio non preceduto da alcun carattere, per indicare una linea vuota

per quanto riguarda l'avvio di una nuova discussione con un utente si seguirà fedelmente la specifica del committente. In questo caso dovremo presupporre quindi un meccanismo che permetta al programma Talk di recuperare l'indirizzo IP dell'utente che si vuole contattare. Si farà uso dello stesso componente sul quale il Listen registra gli utenti che si collegano: chiameremo questo componente name server

Il sistema completo si comporrà di tre elementi:

- **LISTEN**: il software lato server che girerà su una macchina di utente e attenderà le chiamate degli utenti remoti.
- **TALK**: il software lato client che girerà sulla stessa macchina con

possibilità di aprire più istanze. Grazie a quest'ultimo potremo inviare messaggi ad utenti remoti

- **NAME SERVER**: la macchina che si occupa di gestire l'associazione nome-indirizzo IP-porta degli utenti collegati alla chat

Class Diagram (iniziale)

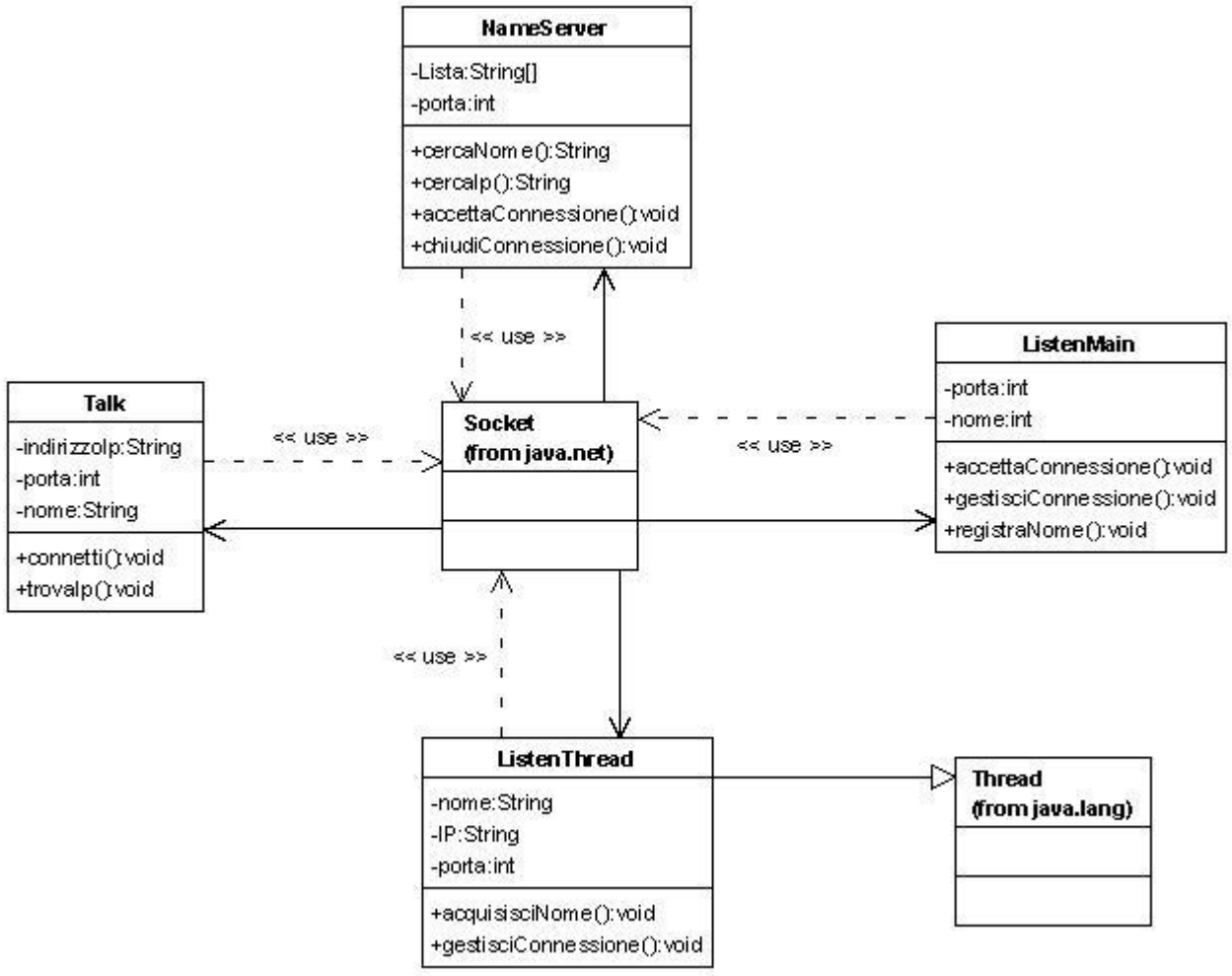

Figura 1: Class diagram iniziale

Analisi dei componenti

A questo punto occorre analizzare i tre componenti fondamentali del programma

LISTEN

Il listen rappresenta il componente che si occupa di ricevere i messaggi, dopo l'analisi dei requisiti si è arrivati alla conclusione che le classi necessarie alla realizzazione del sistema sono tre

- la main_class che si occupa di gestire le connessioni al server, di registrare il nome utente sul name server per evitare ambiguità e di allocare per ogni connessione un opportuno gestore, questa classe in pratica permette l'ascolto delle connessioni su una specifica porta
- la classe listen_thread che gestisce la connessione di un utente remoto. Con tale classe si gestisce il recupero del nome dell'utente a partire dall'indirizzo IP, la ricezione dei messaggi e la gestione della connessione

TALK

Il talk viene realizzato in modo molto semplice visto che deve solamente occuparsi dell'instaurazione della connessione verso un certo utente

Dall'analisi si evince quindi che tale componente necessità di una sola classe:

- la classe Talk che si occupa di inviare messaggi agli utenti remoti, di gestire la connessione e di acuire l'indirizzo IP associato al nome

NAME SERVER

Questo componente si occupa di fornire un servizio che associ gli Ip ai nomi degli utenti e far si che quest'ultimi non siano duplicati. Per renderlo possibile sono stati implementate 2 classi

- la main_class che si occupa di gestire le connessioni al server, e di associare un thread per ogni connessione in arrivo
- la classe server_thread gestisce una singola connessione di un utente remoto. Questa classe rende disponibile 4 servizi, atti a gestire l'associazione Nome – Ip, Porta
    - Connect: registra il nome utente nella lista degli utenti restituendo un errore nel caso in cui il nickname usato sia già registrato
    - Disconnect: elimina dalla lista degli utenti l'entry dell'utente che si vuole

scollegare dal sistema

- retrieveName: restituisce il nome associato ad un indirizzo IP. Questo metodo viene usato dal Listen per acquisire il nome di un utente che si connette
- retrieveIP: restituisce l'IP associato ad un nome. Questo metodo viene usato dal Talk per ricavare l'IP verso cui effettuare la connessione

I tre componenti vanno avviati sulle rispettive macchine con la seguente sintassi:

TALK: java Talk <nome_utente> <indirizzo IP name server>

LISTEN: java Listen <nome_utente> <indirizzo IP name server>

NAME SERVER: java TCPParallelServer [nessun parametro]

In tutti e tre i componenti si è assunto che il servizio chat stia in ascolto sulla porta 7777, mentre quello del name server sulla porta 5555.

Si è qui introdotto un ulteriore parametro <indirizzo IP name server>, nel Talk e nel Listen. Questo parametro permette di settare l'IP del name server, questa scelta, al di fuori delle specifiche, è stata fatta per evitare di implementare un metodo di lettura della configurazione. Tale metodo avrebbe complicato inutilmente l'implementazione. Inoltre il passaggio di tale parametro può facilmente essere eluso usando uno script batch.

Use Case

Lo use case diagram riportato di seguito è molto semplice in quanto per l'applicazione richiesta non si presentano richieste complesse. Nello scenario dell'applicazione si necessita principalmente di due casi d'uso che vedrà come attori uno stesso individuo: nel caso del del ListeningUser sarà un certo utente che si specializzerà nel ruolo di utente che 'ascolta' i messaggi a lui diretti; nel caso del TalkingUser sarà sempre lo stesso utente specializzato stavolta nel ruolo di utente che invia i messaggi. Si è voluta inserire questa distinzione proprio per sottolineare il fatto che si hanno due parti del sistema distinte che possono operare in modo indipendente. Si è anche voluto inserire il ruolo del

name server che in questo caso si presenta come attore atto ad ospitare il componente che gestisce la risoluzione dei nomi. Inoltre si sono inseriti i casi d'uso di connessione e disconnessione relativi all'utente in ascolto per evidenziare la relazione tra l'utente ed il name server al momento dell'avvio del programma. Viene presentato inoltre il caso d'uso del retrieveName e del del retrieveIp asservito al programma stesso.

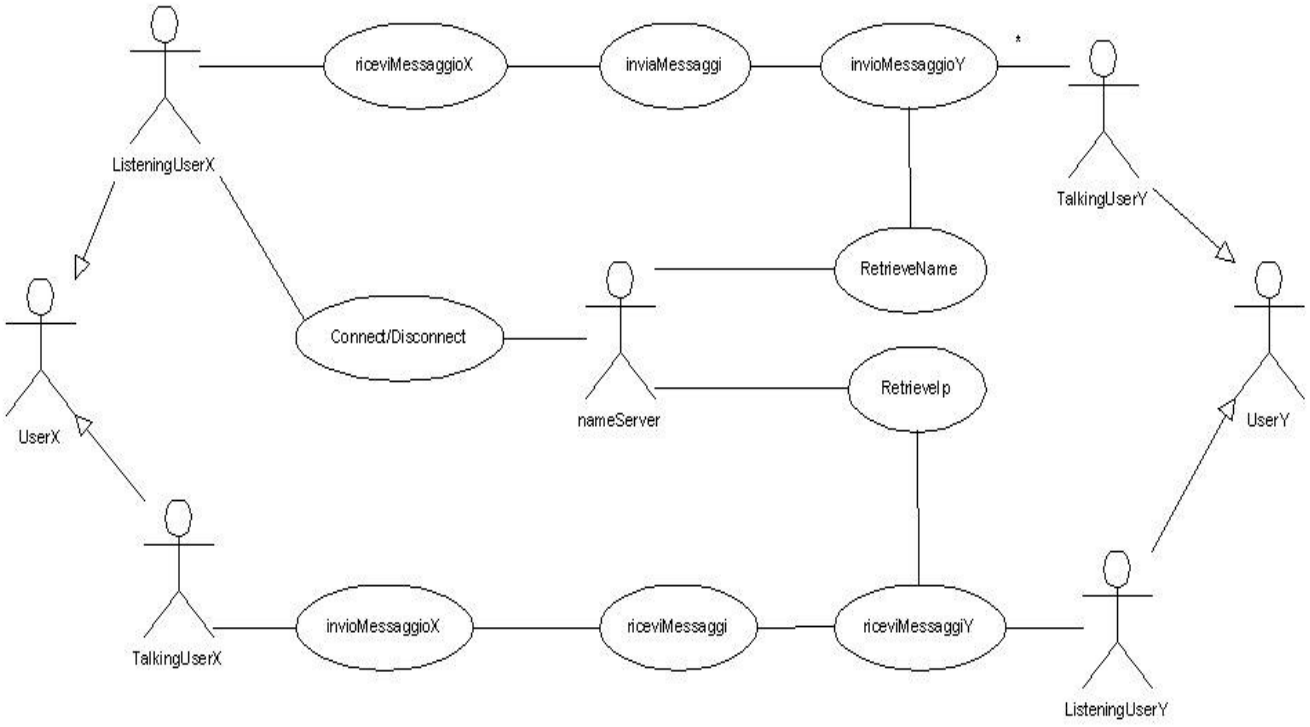

Figura 2: Use case diagram

Architettura software

Il class diagram riportato di seguito è quello dell'implementazione finale. Si procederà adesso con una descrizione delle singole classi:

TCPParallelServer: è una classe che esegue un ciclo while infinito in attesa di connessioni. Ad ogni connessione instaurata l'istanza di questa classe avvia un thread che si occupa dell'effettiva gestione della comunicazione. Questa scelta è stata fatta per permettere la possibilità a più utenti di connettersi simultaneamente senza che gli venga negato l'accesso. Infatti il name server

rappresenta il fulcro del sistema e senza l'accesso ad esso un utente è incapace di usare l'intero sistema. Questa classe inoltra alloca una LinkedList che sarà passata per riferimento ai thread istanziati per permettere di effettuare le operazioni relative ad ogni utente (acquisizione nome o IP, registrazione e deregistrazione).

ServerThread: è la classe che gestisce una connessione d'utente al name server. Viene acquisita dalla socket passata una stringa nel formato <comando>:<parametro>. Tramite l'interpretazione di questa stringa vengono effettuate le operazioni di:

- connessione: viene registrato il nome utente proveniente da "parametro". Quest'operazione può restituire una stringa di errore (identificata da "###<causa errore>") nel caso in cui il nome risulti già registrato.
- disconnessione: viene eliminato dalla lista degli utente on-line, l'utente specificato da "parametro"
- retrieveIP: invia sulla socket l'IP dell'utente "parametro". Restituisce un errore nel caso in cui l'utente sia off-line
- retrieveName: invia sulla socket il nome dell'utente con IP "parametro". Notifica l'assenza dell'utente nel caso in cui l'IP non risulti in elenco.

Talk: è la classe che gestisce l'invio di messaggi verso utenti remoti. Il suo funzionamento si basa sul seguente schema: avvia una connessione verso il server dei nomi, in modo da ottenerel'IP dell'utente. Successivamente instaura una connessione verso tale utente, acquisendo in un while i messaggi. Quando viene inserita una stringa vuota il programma viene chiuso.

Listen: il funzionamento di questa classe è analogo a quello del TCPParallelServer. La prima operazione eseguita è quella di registrare sulname server, il nome specificato da riga di comando. Viene quindi aperta una Socket di tipo server in ascolto delle connessioni degli utenti remoti. Per ogni connessione ricevuta viene quindi istanziato un nuovo ListenThread che gestirà effettivamente la connessione.

ListenThread: è il gestore della connessione dell'utente remoto. La prima operazione eseguita è quella di contattare il name server per acuire il nome associato all'indirizzo IP connesso. Successivamente ricevi i messaggi da parte dell'utente remoto e li stampa a video associandovi il nome dell'utente.

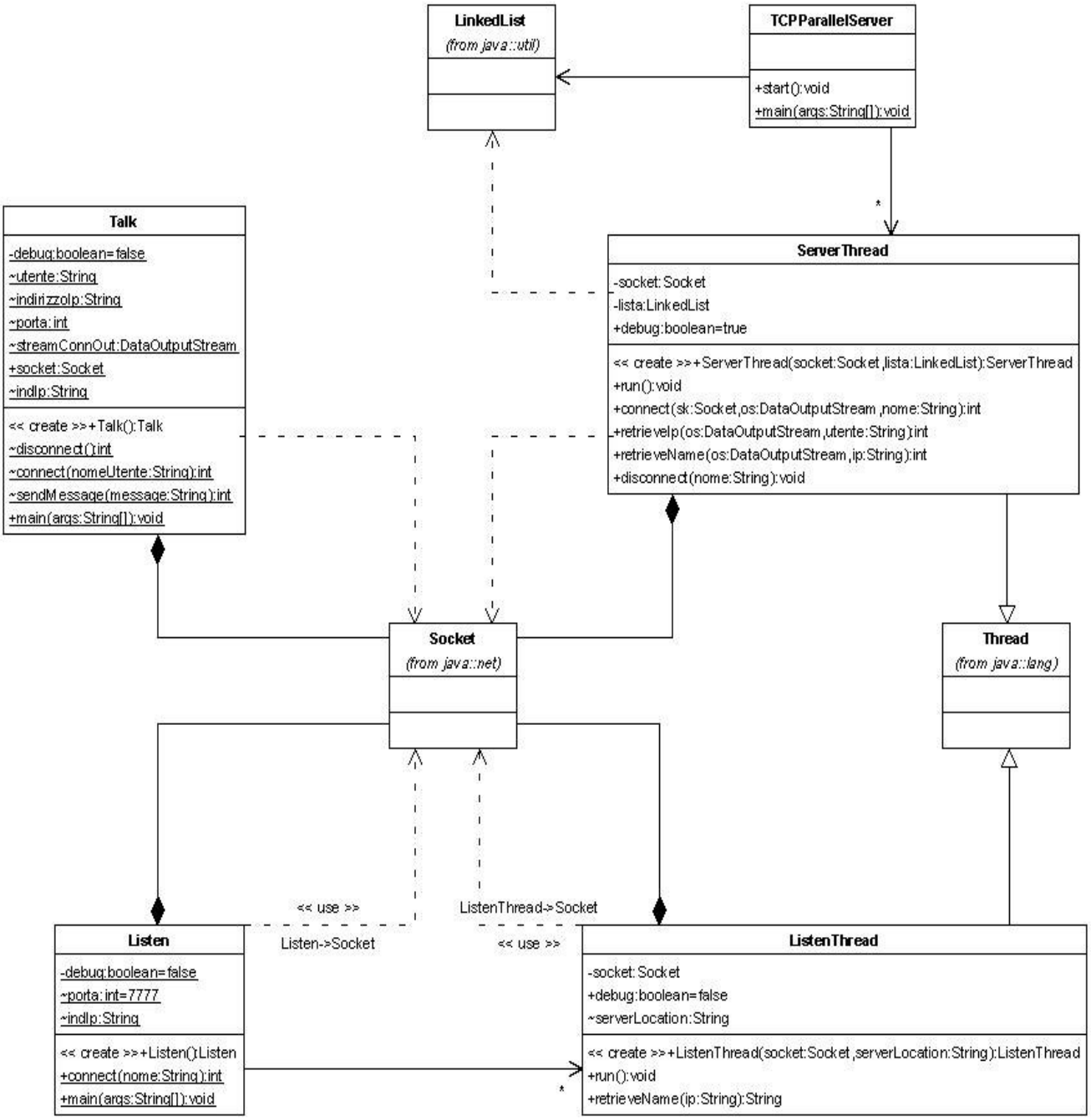

Figura 3: Class diagram completo. Realizzato sulla base dell'implementazione

Sequence Diagram

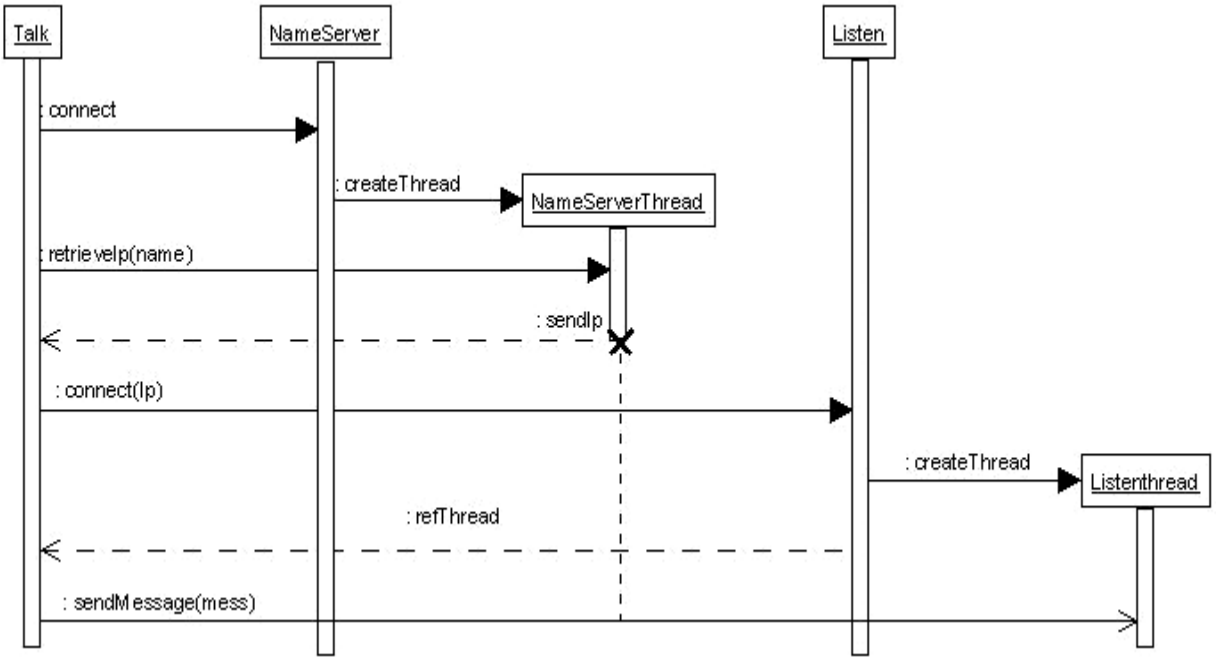

Figura 4 - Sequence Diagram relativo all'invio con successo di un messaggio

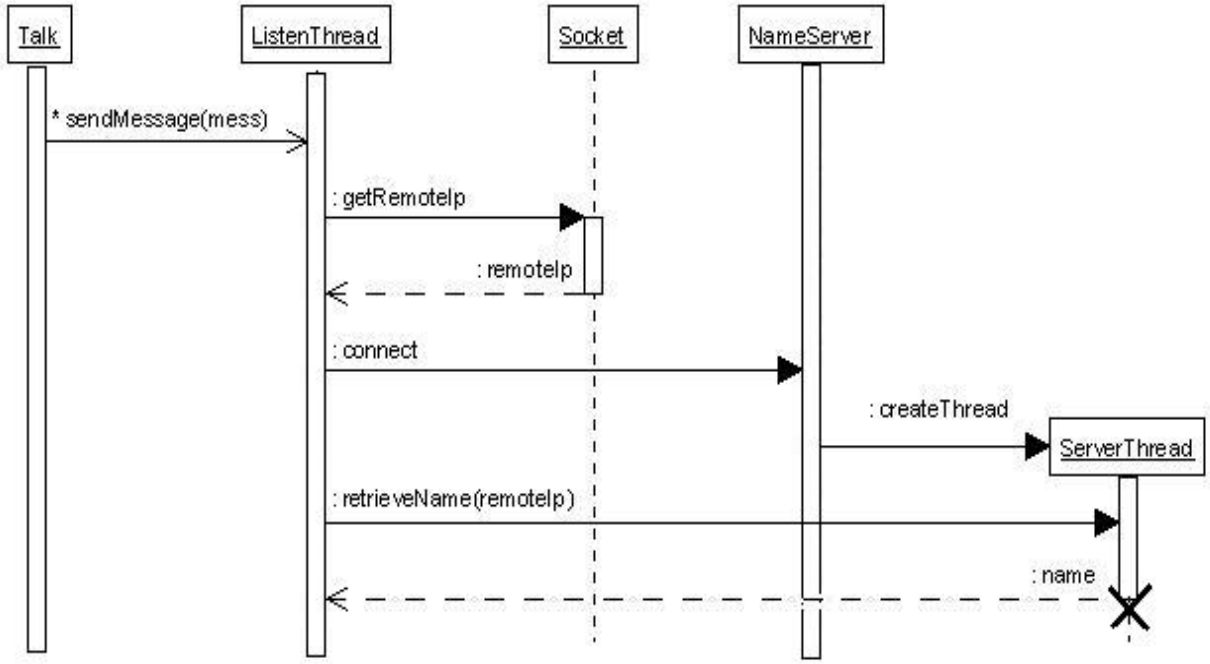

Figura 5 - Sequence Diagram relativo alla ricezione con successo di un messaggio

Activity Diagram

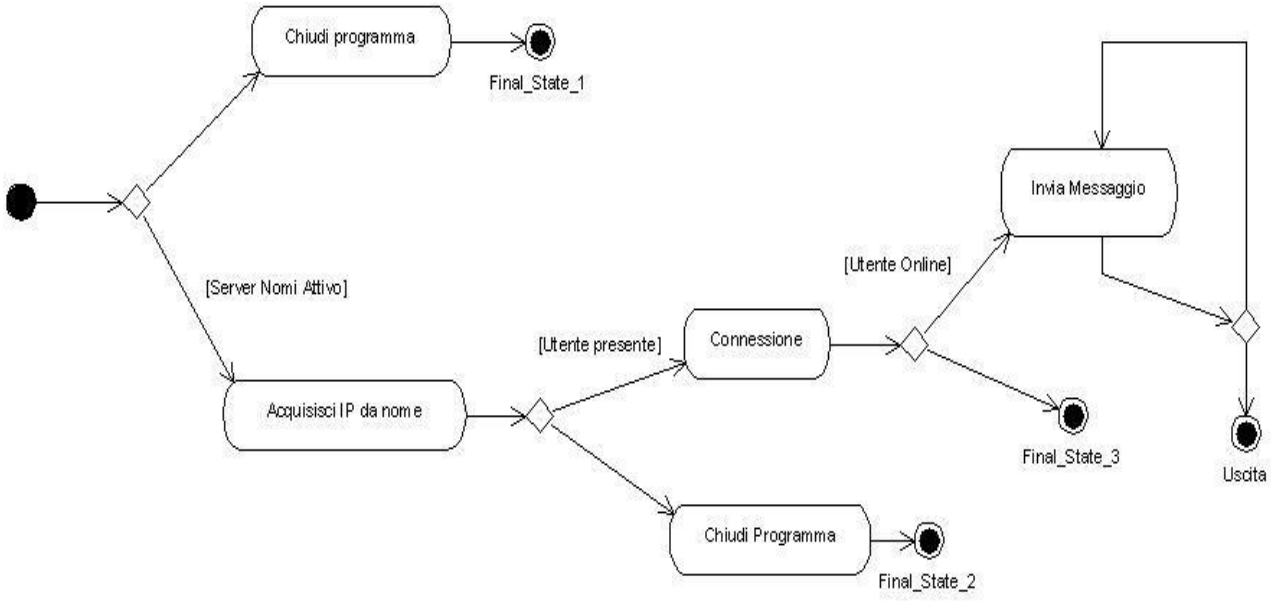

Figura 6 – A.D. relativo all'avvio del Talk. I final-state sono stati che portano alla chiusura del programa

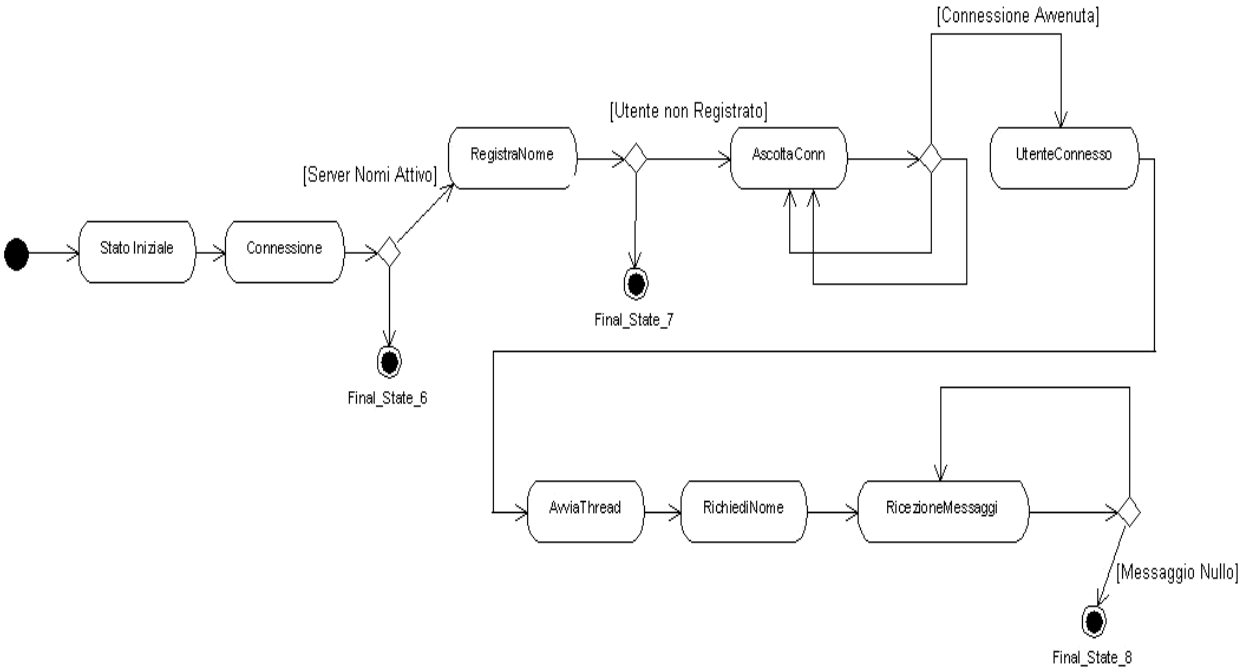

Figura 7 – A.D. relativo all'avvio del Listen e ad una eventuale connessione del programma. Si noti che il programma principale, non viene mai chiuso e rimane nello stato di AscoltaConn

Distribuzione fisica

Il sistema in questione verrà distribuito sulle macchine utente installando i componenti Listen e Talk. Il componente Name Server verrà installato invece su un'unica macchina centrale che abbia un indirizzo IP statico. Quest'ultimo non è un requisito fondamentale, ma desiderabile: dando in questo modo la possibilità agli utenti di usare sempre lo stesso indirizzo IP per localizzare il name server. Il name server in ogni caso non avrà a carico una grossa mole di lavoro per rendere disponibile questo servizio, in quanto gestisce una sola lista di stringhe e le connessioni da parte degli utenti remoti vengono mantenute per pochi secondi al massimo.

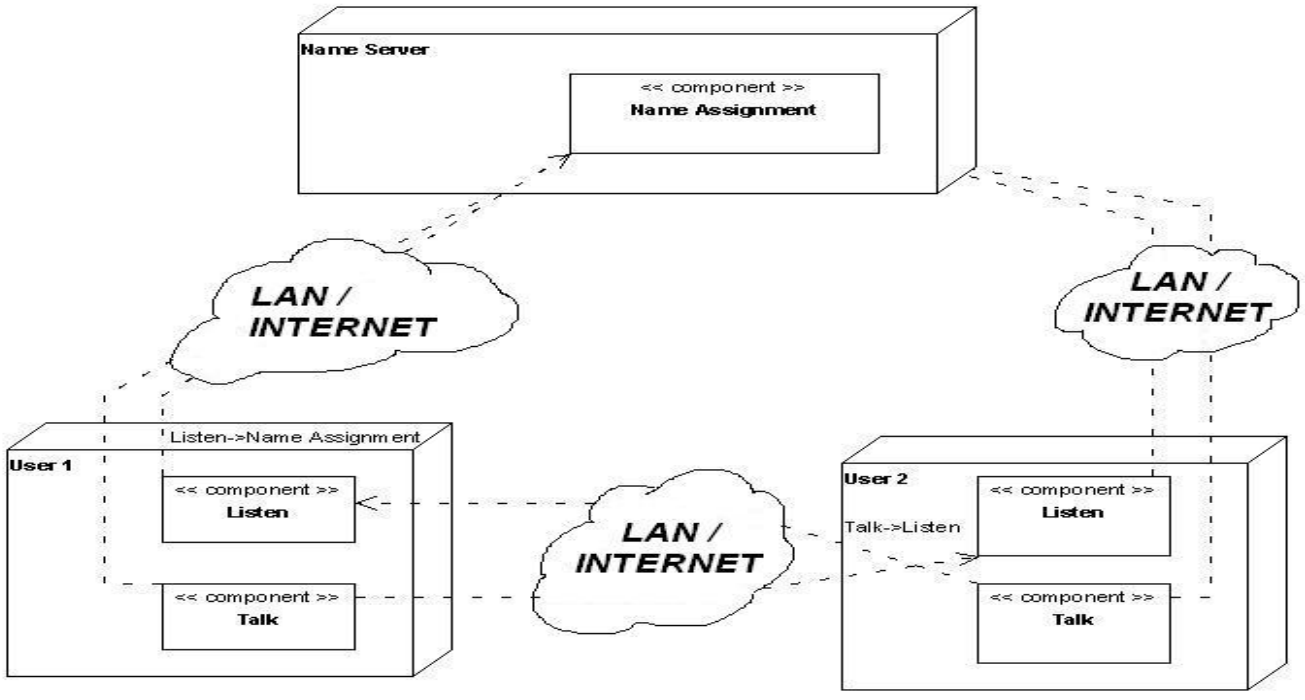

Figura 8: Deployement Diagram

Test

La metodologia per la determinazione degli errori applicata è quella della stampa a video di messaggi e di valori di variabili per verificare il corretto flusso di esecuzione del codice. Si è ritenuto necessario mantenere nel codice finale le istruzioni per la verifica. Per evitare però di "annoiare" l'utente con un eccessivo numero di messaggi si è inserita una variabile di controllo booleana che permette di escludere tali messaggi.

Il sistema è stato testato con diverse modalità:

- LAN: si è creata una rete di tre computer, usandone una come server. Su tutte e tre le macchine è stata poi avviata un'istanza del Listen e 2 del Talk per testare il dialogo contemporaneo di tre utenti
- Internet: sul web è stato fatto un analogo test ma questa volta con 5 macchine avviando su ognuna di esse un'istanza del Listen ed in numero variabile alcune istanze del Talk
- Locale (test preliminari): si è provato il programma anche in locale per verificare che le funzioni principali fossero affidabili e per testare che la gestione degli errori avvenisse nel modo desiderato.

Per ognuno di questi test si è verificato il comportamento dei componenti nei seguenti casi d'errore:

- name server non attivo
- il name server chiude la connessione in modo inatteso
- Listen che cerca di registrare un utente già in linea
- chiusura della connessione del Listen inattesa
- Listen non attivo
- chiusura del Talk in modo inatteso
- tentativo di connessione del Talk verso un utente che non ha eseguito la normale procedura di disconnessione