Progetto di
Reti di Calcolatori

Assegnamento distribuito dei posti al cinema

Autore : Cremonini Davide Matricola: 2148 054902
losceicco@libero.it

Obiettivo:

Realizzare un' applicazione distribuita in grado di soddisfare un' assegnamento concorrente dei posti in un cinema.

Analisi:

La realtà che si intende riprodurre è quella del cinema Stacity di Rastignano (Bo). Si vuole quindi realizzare un'applicazione che preveda l'utilizzo di più stazioni di prenotazione e che sia in grado di assegnare in maniera univoca uno o più posti ai clienti del cinema.
Il progetto rappresenta una realtà leggermente ridotta in quanto prevede l'utilizzo di sole due stazioni di prenotazione (client) e di due server, ma è stato progettato in modo da prevedere una facile estensione sia del numero di client che del numero di server.
Ogni client dovrà comunicare la propria richiesta al server al quale è connesso il quale, a sua volta, prima di dare un'eventuale conferma dell'avvenuta prenotazione, dovrà sincronizzarsi con l'altro server. In questo modo viene garantita una corretta gestione di possibili prenotazioni simultanee.
Inoltre l'utilizzo di due server contemporaneamente attivi ma su nodi diversi conferisce affidabilità e continuità al servizio (ovviamente solo se viene tollerata la possibilità di guasto di uno solo dei due).
Il sistema prevede 5 sale diverse ad ognuna delle quali sono associati tre spettacoli (18,30 – 20,30 – 22,30), proprio come avviene nella realtà quotidiana: questo per dare la possibilità allo spettatore di prenotare in anticipo il posto per una determinata proiezione.
Dopo una prima fase di negoziazione con il server relativo e dopo che si è instaurata con esso una connessione via socket, l'addetto alla distribuzione dei biglietti sceglie la sala e lo spettacolo relativi alle preferenze dell'utente. Tale scelta comporta un'immediata visualizzazione dei posti già occupati per quella proiezione. In questo modo l'utente può visualizzare la disposizione grafica dei posti liberi e scegliere quello che più preferisce. È stata contemplata anche l'ipotesi che l'utente possa cambiare idea sul momento o che l'operatore possa sbagliarsi nell'attribuire i posti richiesti: in questo modo un primo click del mouse sulla casella relativa al posto richiesto comporta la predisposizione all'occupazione del posto stesso, mentre un secondo click riporta il posto nelle condizioni di partenza. Per rappresentare graficamente quanto detto si è scelto di visualizzare la sala attraverso un'insieme di caselle opportunamente ordinate che cambiano colore in base allo stato del posto associato: rosso se occupato, grigio se libero e blu se in fase di assegnamento.
Dopo la scelta del posto, la richiesta viene indtrata al server il quale provvede a verificarne la disponibilità negoziando i dati con l'altro server. Se tutto va a buon fine i posti vengono automaticamente assegnati, le caselle ad essi relative passano da blu a rosso e compare una finestra di avvenuta prenotazione con successo.
In caso di errore invece i posti divengono di colore rosa e un messaggio ci avverte che la prenotazione non è andata a buon fine.

Progetto:

Il progetto è costituito da una serie di files racchiusi all'interno di due package uno per l'applicazione client (StarcityClient) e uno per l'applicazione server (StarcityServer).

StracityClient:

Le classi racchiuse all'interno di questo package sono:

- *ClientIndexGui.java* :interfaccia grafica che interagisce direttamente con l'addetto all'assegnamento dei posti;
- *ClientIndexProtocol.java* :protocollo di comunicazione tra client e server sul quale si appoggia ClientIndexGui;
- *Connection.java* :stabilisce una connessione via socket con il server all'indirizzo IP e alla porta che le sono passati come parametri;
- *ConnectionFrame.java* :interfaccia grafica iniziale che permette all'utente di identificarsi, scegliere la porta e dare il via alla connessione;
- *FrameClientIndex.java* :classe relativa alla costruzione dell'applicazione;
- *FrameErr.java* :finestra che compare alla conclusione di una prenotazione fallita;
- *FrameOk.java* :finestra che compare alla conclusione di una prenotazione avvenuta con successo;
- *Manda.java* :Estensione della classe Thread per mantenere un processo sempre attivo che si occupa di effettuare il dispatching dei messaggi verso il server.
- *Message.java* :Astrazione di messaggio: contiene tutti i costruttori dei messaggi scambiati.
- *MessageBookingThis* :Classe relativa al messaggio di richiesta di prenotazione dei posti da parte del client.
- *MessageGetFreeSeat* :Classe relativa al messaggio di richiesta dei posti già occupati da parte del client.
- *Ricevi.java* :Estensione della classe Thread per mantenere un processo sempre attivo che si occupa di ricevere i messaggi provenienti dal server.

StracityServer:

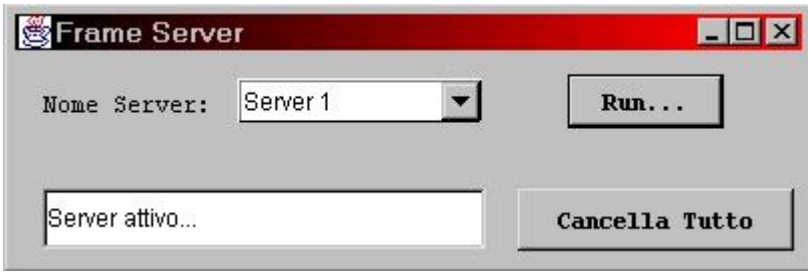
Le classi racchiuse all'interno di questo package sono:

- *ComClient.java* :Estensione della classe Thread che realizza un protocollo relativo alla comunicazione del server con il client;
- *ComServer.java* :Estensione della classe Thread che realizza un protocollo relativo alla comunicazione del server con l'altro server;

- *Connection.java* :stabilisce una connessione via socket con l'altro server all'indirizzo IP e alla porta che le sono passati come parametri;
- *FrameServer.java* :interfaccia grafica iniziale che permette di scegliere la porta di comunicazione con il client, quella con l'altro server e la priorità associata al server stesso;
- *ListenClient.java* :Thread che si mette in ascolto dei client che si vogliono connettere, accetta le connessioni e genera un altro thread (ComClient) che gestisce la comunicazione vera e propria.
- *ListenServer.java* :Thread che si mette in ascolto dei server che si vogliono connettere, accetta le connessioni e genera un altro thread (ComServer)che gestisce la comunicazione vera e propria.
- *Server.java* :Classe relativa alla costruzione dell'applicazione;
- *ServerStdProtocol.java* :Protocollo di comunicazione tra Server e Client.

Funzionamento Server:

Innanzitutto è necessario avviare i due server. L'applicazione è la stessa per entrambi e permette di selezionare il server che si sta attivando. In maniera del tutto trasparente all'utente il programma imposta la porta di comunicazione col client, quella con l'altro server e il numero del server stesso, significativo per indicarne la priorità in caso di assegnamento concorrente. In questo modo si è voluto rendere più agevole l'utilizzo dell'applicazione ad eventuali utenti poco esperti.
Una volta scelto il server si può far partire l'applicazione attraverso la pressione del tasto "Run…"e la scritta "Server attivo…" compare all'interno di una JtextField per indicarne lo stato.
L'interfaccia è la seguente:



FrameServer.java

Ovviamente uno dei due server dovrà funzionare da master e restare quindi in ascolto dell'eventuale connessione dell'altro server: quando questa avviene si crea una connessione via socket tra i due server. Nel caso in esame si è scelto di determinare come predominante il server con indice più basso.

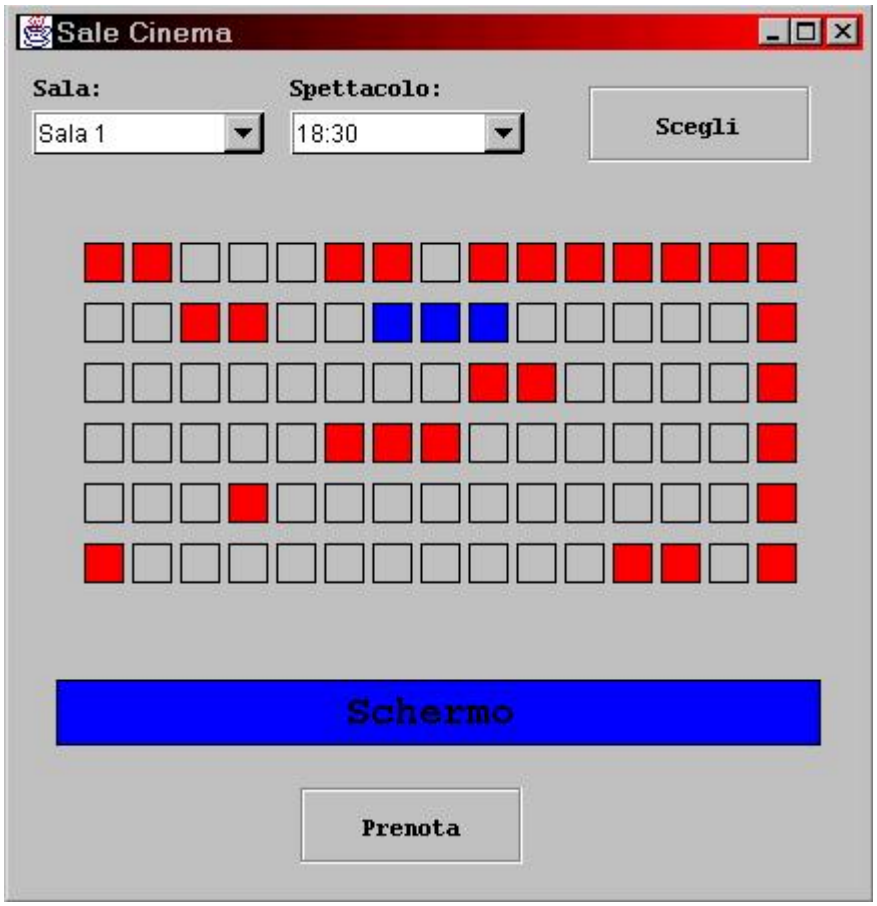
Ogni server presenta anche un tasto "Cancella Tutto" che può essere utilizzato a fine giornata. Questo infatti pulisce tutti i files dalle informazioni che rappresentano gli insiemi dei posti occupati e quindi riporta l'intero sistema server in condizioni di partenza.

Funzionamento Client:

Anche in questo caso l'applicazione realizzata è unica per tutti i client e quindi è necessario impostare alcuni parametri all'avvio. L'interfaccia scelta è molto simile a quella del server e consente sia di impostare il nome dell'operatore associato a quella postazione che l'indirizzo del server al quale ci si vuole collegare.
Sulla base delle considerazioni fatte relative alla facilità d'uso dell'applicazione server, anche questa è strutturata in modo che la scelta del nome dell'operatore determini automaticamente il settaggio del numero della porta di comunicazione col server.
L'interfaccia client è la seguente:

ConnectionFrame.java

Una volta effettuata la scelta è sufficiente premere il tasto connetti per avviare l'applicazione.
La pressione di tale tasto instaura una socket di comunicazione con il server relativo, l'apertura di una nuova finestra associata all' assegnamento dei posti e la chiusura della stessa finestra di connessione. Viene creata in maniera dinamica una matrice di posti rappresentante la mappa della sala in cui ognuno di essi è rappresentato da una casella di testo in grado di cambiare colore in base allo stato. Ovviamente all'inizio sono tutti grigi, poi, a seguito della scelta della sala e dello spettacolo e alla pressione del tasto"Scegli" i posti occupati relativi a quella proiezione vengono automaticamente messi in risalto. Questo perché in seguito alle scelte effettuate il client invia un messaggio di richiesta al server contenente sala e spettacolo il quale risponde con un messaggio contenente l'elenco dei posti già assegnati.
Ora l'operatore può mostrare la mappa allo spettatore e procedere alla scelta dei posti da prenotare. Tale scelta viene effettuata attraverso la selezione della casella corrispondente al posto la quale risponde al click del mouse cambiando colore e passando dal grigio al blu.
Nell'immagine sottostante si può notare la mappa della sala ottenuta a seguito della richiesta dello spettacolo scelto con l'insieme dei posti occupati e quelli in via di prenotazione.
A questo punto è sufficiente premere il tasto "Prenota" per avviare il protocollo che determina se i posti richiesti possono essere assegnati oppure no: il client invia la richiesta contenente l'elenco dei posti al server relativo, questo lo scrive in unfile temporaneo e poi avvia la negoziazione con l'altro server. Se tutto va a buon fine entrambi i server aggiornano il proprio database e viene mandato un messaggio di conferma al client che ha effettuato la richiesta.



ClientIndexGui.java

L'operatore si accorge del successo dell'operazione in quanto questa determina l'apertura di una nuova finestra di dialogo che conferma l'avvenuta prenotazione. Egli non dovrà far altro che confermare e procedere con il cliente successivo.
L'interfaccia grafica di tale finestra è molto semplice:

FrameOk.java

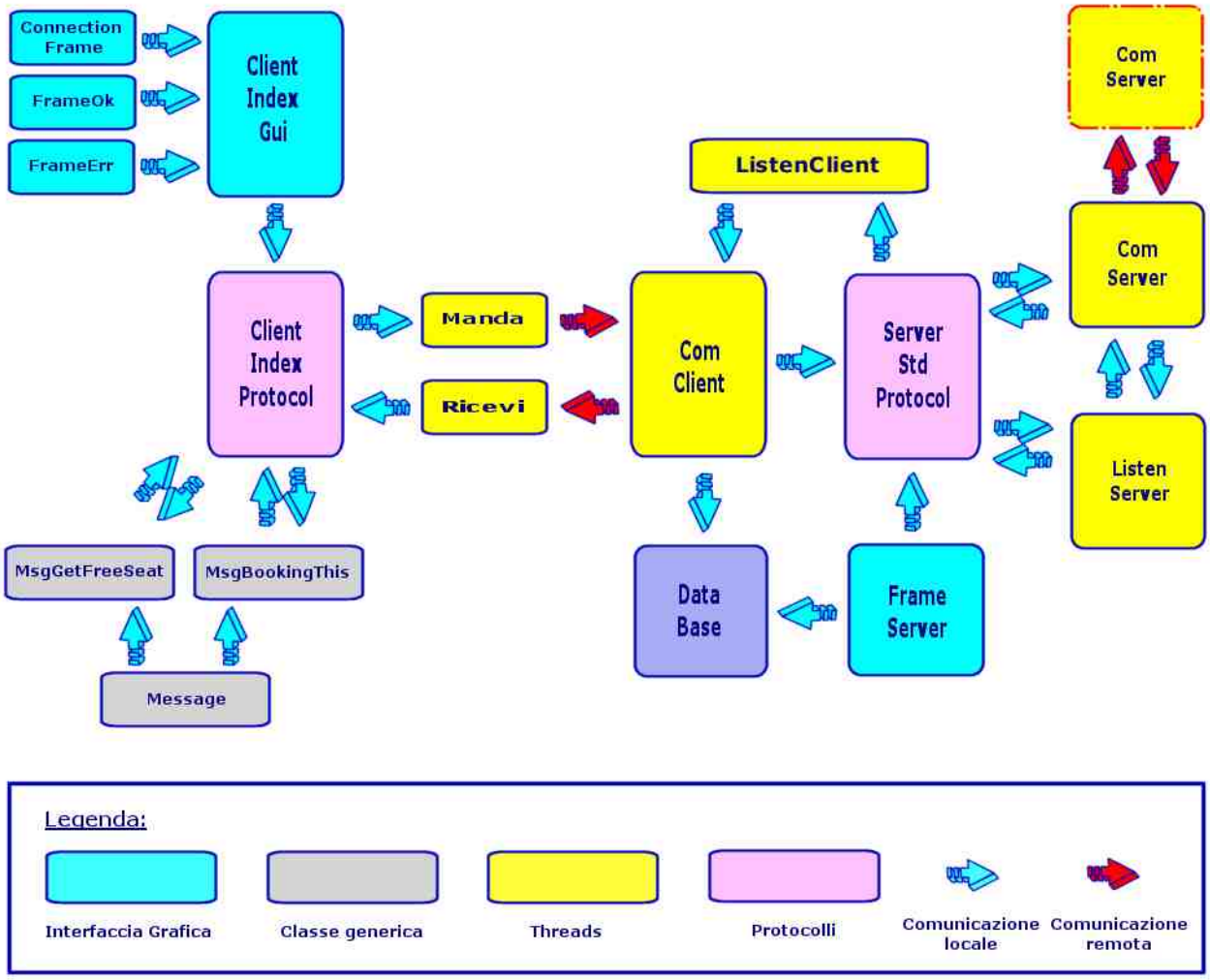
Se invece la prenotazione non va a buon fine in quanto contemporaneamente effettuata dall'altro server, viene visualizzata una finestra di dialogo contenente un messaggio di errore e i posti richiesti passano da blu a rosa. Attraverso la pressione del tasto "Aggiorna" viene ricaricata la mappa dei posti analizzando i posti richiesti: quelli già occupati diventano rossi, mentre gli altri tornano grigi. In questo modo si può effettuare un altro tentativo di prenotazione.

L'interfaccia grafica è la seguente:

FrameErr.java

Implementazione:

L'interazione tra le varie classi è rappresentata nello schema qui sotto:



StarcityClient:

Il frame d'apertura è "***ConnectionFrame***": qui l'operatore sceglie dapprima il proprio nome, poi il server al quale si collega e quindi avvia l'applicazione. La scelta del proprio nome influenza uno **switch** interno il quale associa alla posizione dell'elemento all'interno della comboBox il numero di una porta. In questa maniera è molto facile per l'operatore scegliere le impostazioni iniziali in quanto deve semplicemente selezionare il proprio nome ed è altrettanto facile introdurre nuovi operatori (e quindi nuove porte di comunicazione) in quanto basterà inserire semplicemente nuovi valori all'interno dello **switch**, avendo cura poi di riportarli poi anche nel lato Server.
Anche per quanto riguarda la scelta del server si può procedere il modo analogo anche se per comodità l'associazione numero/indirizzo operata dallo **switch** in questo caso viene fatta solo in seguito, ossia all'interno della classe **ClientIndexGui**. Di questa ne viene creata un'istanza in seguito alla pressione del tasto connetti. I parametri passati sono il numero della porta e il valore del server appena scelti.
Lo scopo di questa classe è quello di rendere più agevole l'utilizzo dell'applicazione. All'atto della creazione vengono create due comboBox che mi permetteranno di inserire l'ora dello spettacolo e la proiezione (rappresentate in termini di interi rispetto alla loro posizione all'interno della comboBox) e subito sotto viene rappresentata la sala. Questa è costituita da una matrice di **JTextArea** di dimensioni prefissate. Anche in questo caso la scelta effettuata consente eventualmente di impostare il numero di posti (quindi di file e di colonne) attraverso un opportuno settaggio delle costanti **NUMROWS** e **NUMCOLS**. Le impostazioni di ogni **JTextArea** vengono settate attraverso un doppio ciclo in modo da percorrere tutti gli elementi della matrice. Questa classe comunica direttamente con la classe **ClientIndexProtocol** e già all'inizio ne viene creata un'istanza; oltre alla porta e all'indirizzo del server la classe in questione passa anche un riferimento a se stessa in quanto una volta ricevuta la risposta relativa allo stato dei posti dovrà occuparsi di rappresentarne lo stato.
La classe **ClientIndexProtocol** racchiude tutto il protocollo di comunicazione tra client e server: all'atto della creazione di una sua istanza vengono immediatamente creati dei flussi di input e output per poter leggere e scrivere direttamente sulla socket; contiene un metodo **procedi()** che è direttamente invocato dalla pressione del tasto "**scegli**" il quale provvede a creare ed avviare le istanze dei thread per la comunicazione "Manda" e "Ricevi". Quindi viene creata un'istanza di **MsgGetFreeSeat** in modo da poterne poi utilizzare la variabile di classe **outMsg** combinata al metodo **mandaMSG** del thread **manda** per poter inviare al server la richiesta dei posti già occupati per la proiezione scelta. L'utilizzo della classe MsgGetFreeSeat permette di incapsulare il tutto all'interno di una classe. La richiesta sarà formulata nel seguente modo: [NomeMsg | NumSala | NumSpettacolo]. Per esempio: "GFS11". Quindi il Client si mette in attesa di una risposta da parte del Server. Risposta che viene ricevuta attraverso il metodo **RiceviMSG()** del thread **Ricevi** e passata immediatamente al all'oggetto ClientIndexGui che provvederà a trasformarla in forma grafica. Il metodo procedi termina qui.
L'istanza della classe ClientIndexGui però contiene un secondo pulsante etichettato con Prenota, alla pressione del quale viene richiamato il metodo **prenota()** dell'oggetto ClientIndexProtocol. Questo metodo si preoccupa di ricavare l'elenco dei posti per i quali si richiede la prenotazione e di creare una nuova istanza dell'oggetto **MsgBookingThis** in modo da poter sfruttar nuovamente il metodo **mandaMSG** del thread **Manda** per formulare la richiesta al server. Il messaggio assumerà la forma: [NomeMsg | Sala | Spettacolo | ListaPosti]. Per esempio: "BTH11 0 9 0 10". Anche in questo caso la ricezione della

risposta è affidata al thread Ricevi mentre il protocollo si occuperà di riconoscere se la prenotazione è andata a buon fine oppure no. In caso affermativo si procederà alla colorazione dei posti di rosso, altrimenti di rosa. Inoltre, sempre in base all'esito della prenotazione, l'oggetto ClientIndexProtocol si preoccuperà di aprire le finestre di dialogo **"FrameOK"** oppure **"FrameErr"** per darne comunicazione all'utente.

StarcityServer:

Per quanto riguarda la parte server, invece, l'applicazione di partenza è **FrameServer**: questa mi permette di scegliere il server da avviare. A seguito della pressione del pulsante **"Run"** la scritta **"server attivo…"** compare nella JTextField sottostante per indicare che il server è in funzione e viene generata una nuova istanza della classe ServerStdProtocol al quale viene passato come parametro il numero del server. Analogamente a quanto si verifica nel client, all'interno di questa classe viene eseguito uno switch che in base al numero di server scelto provvede automaticamente a settare il numero di porta per la comunicazione col client relativo e quella per la comunicazione con l'altro server in modo del tutto trasparente all'utente. Qui però i problemi aumentano, soprattutto perché devo riconoscere se un server funzionerà poi da client o da server nei confronti del server col quale è collegato. Nel caso in esame ho scelto di associare al server con il numero più basso la funzione di server mentre all'altro quella di client. A tale proposito il server avviato deve conoscere la sua condizione in modo da potersi mettere in ascolto di un'eventuale connessione da parte di un altro server oppure di farne direttamente richiesta. Quindi se per esempio il server scelto è il numero 1 viene creata un'istanza della classe ListenServer la quale dapprima si mette in ascolto sulla porta di comunicazione con l'altro server per un'eventuale connessione e poi, quando questa si verifica, genera un thread chiamato ComServer che si occuperà invece della comunicazione vera e propria. Come si vede dallo schema precedente, entrambe queste ultime dovranno comunicare con le classi che le hanno generate , quindi, tra i parametri passati, ci dovrà essere anche un riferimento alla classe madre. Se invece il server in questione funziona da client verso l'altro server allora cerco subito di stabilire una connessione con questo generando un'istanza della classe ComServer. Entrambi comunque dovranno mettersi in ascolto di eventuali client che voglionoconnettersi. A tale proposito genero un'istanza della classe ListenClient la quale attende la richiesta di connessione da parte di un client. Quando ciò avviene genero un nuovo thread, istanza della classe ComClient, il quale si occupa della comunicazione vera e propria liberando il server da tale onere. E proprio questa classe associa dei flussi di input e output alla socket di comunicazione. Quindi innanzitutto dovrà ricevere un messaggio contenente la richiesta dei posti già occupati relativamente ad un determinato spettacolo, quindi dovrà estrapolarne il numero della sala, dello spettacolo e andare a prelevare le informazioni dal database locale. Le informazioni ricavate vengono trasformate in messaggio e quindi spedite al client. Quindi il server si aspetterà la richiesta di prenotazione di determinati posti per quello spettacolo. Quando questo messaggio arriva da parte del client la classe ComServer la scrive dapprima su un file locale denominato **"temp.dat"** e quindi ne invia richiesta di conferma all'altro server attraverso un metodo chiamato **"verifica"**. Questo metodo riceve come parametro di ingresso il messaggio appena ricevuto dal client contenente la richiesta di occupazione dei posti relativo ad una determinata sala e ad un determinato spettacolo. Sarà la classe ComServer dell'altro server a gestire il messaggio per la conferma dell'assegnamento. Innanzitutto dovrà estrapolare sala e spettacolo e

quindi verificarne la disponibilità. Dovrà anche verificare che tale posto non sia già in fase di assegnamento per il proprio client per evitare di effettuare una doppia prenotazione per lo stesso posto. È per questo motivo che il messaggio di richiesta viene preceduto da un numero indicante la priorità del server che ne ha fatta richiesta. Priorità che coincide con il numero del server. In questo modo per esempio il server 1 sarà più prioritario del server 2. Quindi se il server 1 riceve dal server 2 la richiesta di prenotazione di un posto che però anche lui sta prenotando, a seguito del confronto di priorità invierà al server 2 una risposta di mancata prenotazione. Accadrà il contrario se il ruolo dei due server dovesse essere invertito.
Ricapitolando quindi, da una parte la classe ComServer si occuperà di costruire il messaggio di richiesta inserendo la priorità del server associato, di spedire il messaggio all'altro server, di riceverne risposta e di reindirizzarla a ComClient, mentre dall'altra parte la classe ComServer si occuperà di ricevere il messaggio, effettuare gli opportuni controlli e di spedire la risposta al server. Tale risposta conterrà la stringa "OK" in caso di successo, mentre la stringa "NO" seguita dai posti già in fase di prenotazione in caso di fallimento. Nel primo caso ovviamente verrà aggiornato il database locale in modo da avere per entrambi i server una copia sempre aggiornata della disposizione dei posti. La classe ComClient quindi dovrà poi comportarsi di conseguenza in seguito al tipo di messaggio ricevuto. Se la prenotazione va a buon fine si dovrà occupare di inviare una risposta di conferma al proprio client, altrimenti invierà errore e il client potrà aggiornare l'elenco dei posti già occupati. A termine dell'operazione, sia che essa sia andata a buon fine sia che sia fallita, il sistema si riportain condizioni di partenza ed è pronto per soddisfare un'ulteriore richiesta.

Prove effettuate:

Sono state effettuate alcune prove per verificare il funzionamento dell'applicazione.

Prova n° 1:

La prima si riferisce al caso più generale, ossia all'assegnamento con successo di una serie di posti richiesti.
Per poter eseguire l'applicazione in locale bisogna procedere nel seguente modo:

1. Avviare l'applicazione server "FrameServer.class", scegliere come server il "Server 1" e avviarlo premendo il taso "Run";
2. Ripetere quanto riportato al passo 1 ma scegliendo questa volta il "server 2".
3. Avviare l'appicazione "ConnectionFrame.class", scegliere come operatore "Davide", come indirizzo del server "localhost" e premere il tasto "Connetti".
4. Si aprirà una nuova finestra contenente due combo box, attraverso le quali scegliere sala e spettacolo, e una mappa contenente l'insieme dei posti (ovviamente per ora tutti liberi).
5. Si procede scegliendo sala e spettacolo e premendo il tasto "Scegli"; automaticamente la mappa dei posti mostrerà quelli già occupati.
6. Si scelgono i posti che si vogliono assegnare;
7. Si simula un errore per verificare l'esatto comportamento dell'interpretazione degli eventi del mouse: basta riclickare su un posto scelto per farlo cambiare di stato e sceglierne eventualmente un altro;

8. Si preme il pulsante "prenota": dopo poco compare una finestra di dialogo che ci informa dell'avvenuta prenotazione. Confermando ci ritroviamo nella finestra precedente.

Prova n° 2:

Si procede come per la prova 1 ma questa volta si sceglie l'operatore "Luigi": in questo modo non ci si collega più al server 1 ma al server 2 e si può verificare se l'operazione risulta esattamente simmetrica.

Prova n° 3:

Con questa prova si vuole verificare il corretto funzionamento in caso di assegnamento concorrente. Per fare questo bisogna procedere nel seguente modo:

1. Si procede come per la prova 1 fino al punto 7, ma questa volta si sceglie come operatore "Luigi". Infatti visto che in questo caso Luigi si connette al server meno prioritario la prenotazione non dovrebbe andare a buon fine.
2. Ora, prima di procedere alla richiesta di assegnamento, si apre "manualmente" il file "temp.dat" presente dentro **"remoteFiles/"** e vi si scrive una stringa contenente un possibile messaggio che riporti almeno uno dei posti scelti per la prenotazione (per esempio "BTH12 0 0", dove BTH è il nome del messaggio, 1 indica la sala 1, 2 indica lo spettacolo delle 20:30 e la coppia 0 0 indica che il posto scelto è quello relativo alla posizione 00). Si salva il file.
3. Si procede alla richiesta premendo il tasto prenota.
4. L'applicazione risponde con una finestra di dialogo riportante un messaggio di errore e l'insieme dei posti richiesti sulla griglia passano da blu a rosa. Alla pressione del tasto "Aggiorna" i posti che tra quelli scelti risultano già prenotati diventano rossi, gli altri tornano grigi.

Prova n° 4:

Si è provato anche a simulare una situazione più vicina alla realtà procedendo nel seguente modo:

1. Si procede all'assegnamento di un posto analogamente a prima;
2. Si presuppone che lo spettatore successivo non voglia assistere allo stesso spettacolo ma ad un altro, quindi si tenta di cambiare sala e spettacolo e si preme il tasto "Scegli";
3. La mappa relativa alla proiezione scelta viene caricata e cambia quindi la disposizione dei posti occupati.
4. Si procede all'assegnamento dei posti in maniera classica.