

# Implementazione di una Chat-Quiz basata su spazi di tuple

Rossella Rubino

## Abstract

La chat è un'applicazione, molto diffusa su Internet, che riunisce contemporaneamente due o più utenti in stanze (*room*) virtuali allo scopo di intrattenere discussioni. Questo progetto si occupa di realizzare una chat-quiz, con l'obiettivo di analizzare l'approccio che sfrutta gli spazi di tuple come modello di coordinazione. Durante la realizzazione di tale progetto sono stati esaminati gli aspetti chiave dei sistemi distribuiti come la replicazione, la comunicazione e la sincronizzazione.

## 1 Introduzione

L'obiettivo di questo progetto è stato realizzare una chat sfruttando come modello di coordinazione gli spazi di tuple. La chat è arricchita, inoltre, dalla possibilità di partecipare ad un quiz distribuito.

Nella parte che segue verranno illustrati i momenti chiave del processo di realizzazione della chat-quiz. La sezione 2 descrive i requisiti funzionali e non funzionali dell'applicazione. Nella sezione 3 verrà illustrata l'architettura dell'applicazione e il protocollo di comunicazione. La sezione 4 descrive come è stato implementato il prototipo e infine nella sezione 5 si conclude discutendo dei futuri sviluppi.

## 2 Specifica dei requisiti

La chat è un'applicazione, molto diffusa su Internet, che riunisce contemporaneamente due o più utenti in stanze (*room*) virtuali allo scopo di intrattenere discussioni. I requisiti funzionali di base per tale progetto risultano essere tre; ogni utente, nella chat, deve, tramite un'interfaccia grafica, poter:

- entrare
- uscire
- inviare msg agli altri utenti

Accanto a tali funzionalità, la chat realizzata in questo progetto, offre anche la possibilità di partecipare ad un quiz che consentirà agli utenti di incrementare il loro punteggio. Esistono, pertanto, altri requisiti che devono consentire ad un utente di:

- partecipare al quiz
- rispondere ai quesiti
- lasciare il quiz

Oltre ai requisiti funzionali, propri del progetto, sono presenti anche requisiti non funzionali legati alle prestazioni, all'affidabilità e alla portabilità del sistema.

### 3 Analisi

L'analisi verrà schematizzata presentando casi d'uso e scenari seguendo lo standard UML, un linguaggio utile per specificare, visualizzare e realizzare sistemi SW anche di grandi dimensioni [3].

#### 3.1 casi d'uso

I casi d'uso sono uno strumento utile per analizzare le interazioni dell'utente con l'applicazione e le azioni che ne scaturiscono.

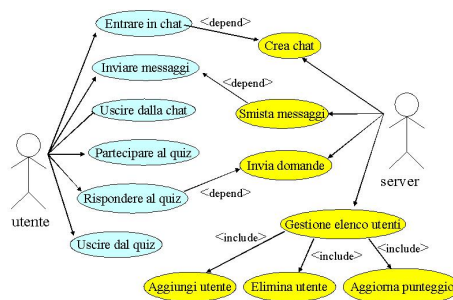


Figura 1: Casi d'uso

### 3.2 scenari

Gli scenari permettono di analizzare in modo più dettagliato i casi d'uso che si ritiene siano significativi per esporre le funzionalità del sistema. Di seguito, per brevità, ne vengono presentati solo due.

<b>Scenario1</b>
<b>Scopo:</b> inviare messaggi nella chat
<b>Attori:</b> utente
<b>Precondizioni:</b> server attivo
<b>Postcondizioni:</b> nuovo messaggio per gli utenti della chat
<b>Scenario principale:</b> - l'utente scrive il messaggio; - il messaggio viene visualizzato nella sua finestra e in quella di tutti gli utenti

Tabella 1: Scenario di invio messaggi

<b>Scenario2</b>
<b>Scopo:</b> rispondere al quiz
<b>Attori:</b> utente, server
<b>Precondizioni:</b> l'utente ha ricevuto una domanda
<b>Postcondizioni:</b> -
<b>Scenario principale:</b> - l'utente invia la risposta come se fosse un messaggio; 2a. se la risposta è corretta, riceve un messaggio che gli comunica che ha vinto e il nuovo punteggio; 2b. se la risposta è sbagliata non riceve alcuna comunicazione

Tabella 2: Scenario di invio risposta al quiz

## 4 Progetto

La prima fase del processo di progettazione consiste nel determinare il modello che si vuole adottare. Le scelte possibili sono

- **modello C/S:** caratterizzato dalla presenza, tipicamente, di un server che implementa un determinato servizio; riceve le richieste da più client e conseguentemente fornisce le risposte;

- **modello a contenimento**: l'applicazione viene integrata in un ambiente che si occupa in modo autonomo di alcuni compiti, di cui è alleggerito il progettista, evitando che si verifichino errori e controllando eventuali eventi [1];
- **modello a tuple**: il nucleo di questo modello è lo spazio di tuple, un insieme strutturato di relazioni, intese come contenitori di attributi e valori. Sullo spazio di tuple è possibile depositare/estrarre informazioni senza limiti di spazio, tempo e senza possibilità di causare interferenze [1].

Per questo progetto si è scelto di adottare un modello ibrido tuple/client-server. Il modello a tuple perchè realizza comunicazione e sincronizzazione senza che sia necessaria la mutua conoscenza tra i processi coinvolti. Viene, inoltre, garantita in tal modo una certa qualità del sistema dovuta alla persistenza degli spazi di tuple e all'assenza di interferenze.

Tuttavia la chat è anche arricchita da un quiz che necessita di un gestore e quindi le entità che intervengono nel processo di comunicazione possono essere distinte in due tipi:

- *agenti* - utenti della chat
- *server* - gestisce la comunicazione e il quiz

L'applicazione è stata così progettata: ad ogni utente è associato uno spazio di tuple per la ricezione dei messaggi relativi alla chat e per i quesiti del quiz. Esistono inoltre altri tre spazi di tuple gestiti dal server, uno per la chat e due per il quiz. Quello per la chat funge da punto di sincronizzazione oltre che da deposito delle richieste inoltrate dagli utenti. Per il quiz, invece, si è ritenuto necessario creare due spazi di tuple: uno che contenesse l'elenco dei partecipanti con il rispettivo punteggio e uno per l'elenco delle possibili domande.

In alternativa si poteva utilizzare un unico spazio di tuple per tutto o eliminare solo gli spazi di tuple associati ai singoli utenti alleggerendo così il progetto del server ma questo avrebbe portato ad una minor chiarezza dell'architettura dell'applicazione e avrebbe costretto i client a preoccuparsi di ulteriori fattori riducendo così la trasparenza.

## 4.1 Protocollo di comunicazione

La comunicazione tra le diverse entità avviene tramite gli spazi di tuple sui quali sono definite le seguenti primitive:

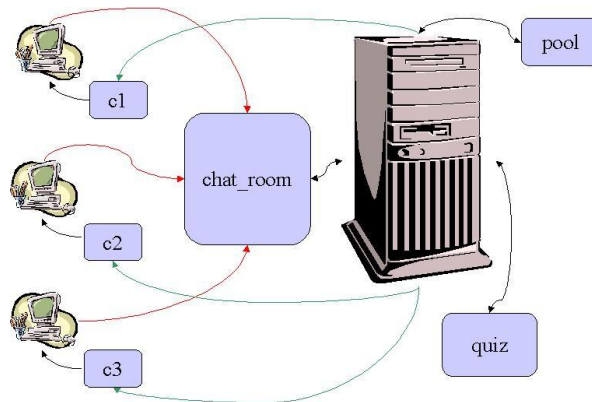


Figura 2: schema chat-quiz

- **in**: preleva la tupla dallo spazio di tuple;
- **rd**: legge la tupla senza estrarla;
- **out**: deposita una tupla

Le tuple adottate in questo progetto sono tuple logiche cioè strutture dati costituite da un nome e da una lista di argomenti.

Esempio:

```
msg(from(Bob),text(ciao))
```

Lo startup del sistema avviene con l'avvio del server. Quando viene attivato un client viene depositata nello spazio di tuple **chat\_room** la richiesta di partecipazione. Il server deve verificare che l'utente abbia un nome univoco, controllando i nomi di tutti gli utenti attualmente presenti nella chat, e comunicare l'esito dell'indagine depositando una tupla nello spazio di tuple del cliente. Se l'esito è positivo l'utente potrà partecipare alla chat altrimenti dovrà ritentare con un nome diverso (Fig.3).

Una volta entrato, il client, potrà inviare messaggi agli altri utenti o uscire dalla chat. Queste azioni avvengono depositando apposite tuple nello spazio di tuple **chat\_room**. Sarà poi compito del server prelevarle e depositarle negli spazi di tuple dei singoli utenti o compiere altre azioni di gestione del sistema (Fig. 4).

Il server si occupa, inoltre, del quiz e quindi dell'elenco dei partecipanti con il loro punteggio e dell'invio a questi delle domande (*multicast*). I client comunicano al server che intendono partecipare al quiz, anche in questo caso, depositando un'apposita tupla nello spazio di tuple **chat\_room** (Fig 5) e

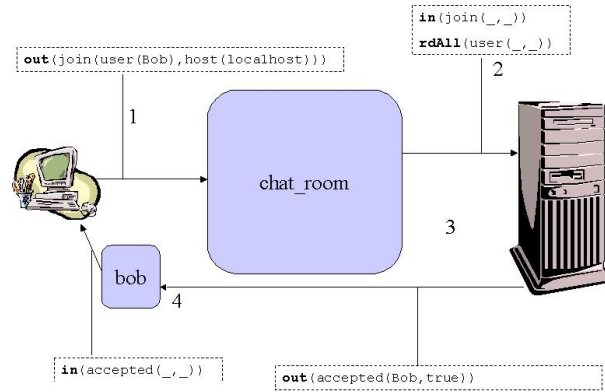


Figura 3: protocollo per l'accesso alla chat

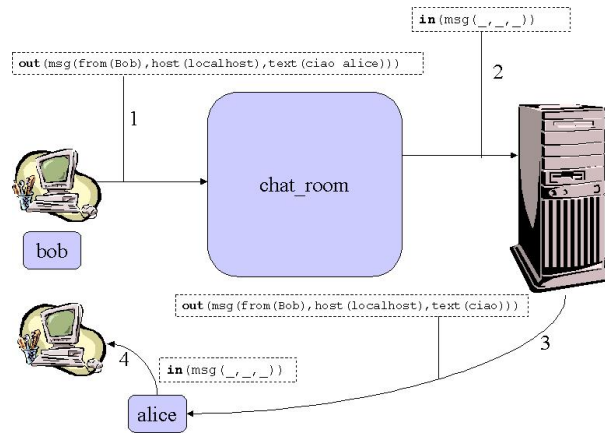


Figura 4: protocollo per l'invio di messaggi nella chat

attendono le domande (continuando a chattare) nel loro spazio di tuple. Il server mantiene l'elenco nello spazio di tuple `quiz` e preleva le domande da `pool` (Fig 6).

Quando un client risponde correttamente al quiz il suo punteggio viene incrementato e l'evento gli viene notificato nel suo spazio di tuple (Fig 7). Nel momento in cui un utente non intende ricevere più i messaggi relativi deve comunicarlo al server sempre depositando una determinata tupla nello spazio di tuple `chat_room`.

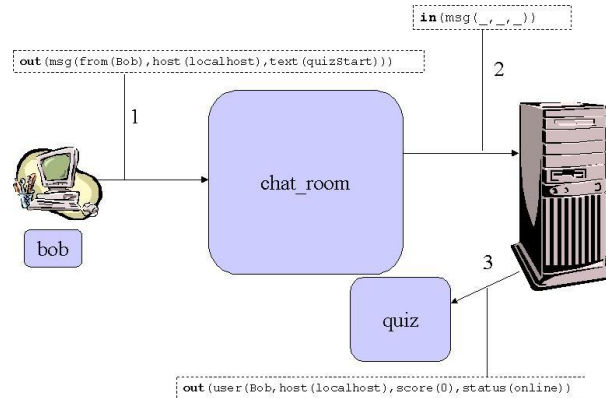


Figura 5: protocollo per la richiesta di partecipazione al quiz

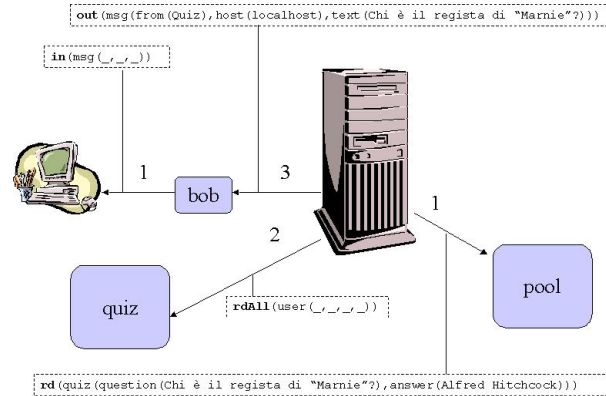


Figura 6: invio di domande ai partecipanti al quiz

## 4.2 Replicazione

Gli spazi di tuple offrono un primo grado di affidabilità perchè la loro locazione è indipendente da quella di client e server e quindi in caso di un loro guasto nessuna informazione andrà persa e il sistema continuerà a funzionare senza alcuna azione di recovery. Per prevenire, invece, l'ipotesi che vada in crash il nodo su cui risiedono gli spazi di tuple le informazioni in essi contenute vengono memorizzate anche in un secondo nodo, il cui stato viene sempre mantenuto aggiornato (modello di replicazione a *copie calde*[1]). L'ipotesi che si fa è di guasto singolo, cioè si ipotizza che possa andare in crash solo uno dei due nodi. La replicazione riguarda solo il nodo che contiene gli spazi di tuple `chat_room`, `quiz`, e `pool` poichè le informazioni contenute negli spazi di tuple degli utenti sono temporanee e quindi legate al tempo di

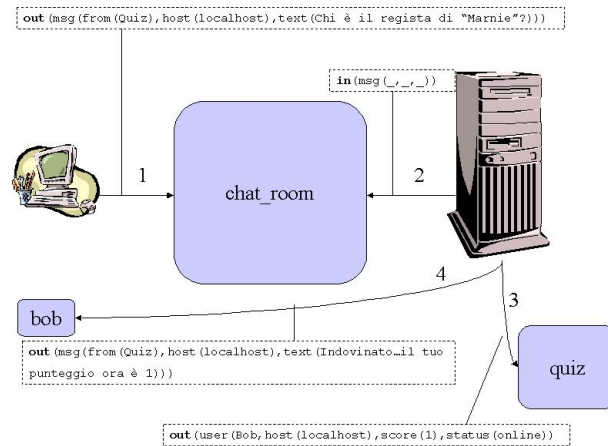


Figura 7: attribuzione punteggio quiz

vita dell'utente. Si ritiene che un grado di replicazione superiore sia superfluo per l'applicazione e che aumenti i costi sia in termini di spazio sia di tempo incidendo così anche sull'efficienza (*Principio di minima intrusione [1]*).

Un'ulteriore ipotesi prevede che quando un nodo va in crash e viene riparato non rientri immediatamente ma solo quando è stata ripristinata la coerenza tra i due nodi; è necessario cioè sospendere il servizio.

## 5 Implementazione

Al fine di presentare le funzionalità della chat-quiz è stato realizzato un prototipo in Java basato sull'infrastruttura di coordinazione Tucson.

Il linguaggio di programmazione Java è stato scelto per la sua portabilità mentre Tucson perché offre degli spazi di tuple programmabili (*centri di tuple*) cioè arricchiti da una specifica di comportamento; programmando tale comportamento si possono realizzare le politiche desiderate[2]. Nonostante i vantaggi offerti da Tucson, in questo progetto non si è sempre sfruttata appieno la programmabilità dei centri di tuple perché l'obiettivo era, semplicemente, analizzare l'approccio che sfrutta il modello a tuple per la comunicazione e la sincronizzazione.

Nella figura 8 viene mostrato il diagramma delle classi che compongono l'applicazione. La classe **ChatAgent** realizza un'interfaccia grafica che permette agli utenti di inviare messaggi e visualizzare la chat (Fig. 9). Uno o più utenti sono associati ad un server implementato dalla classe **ChatServer** che



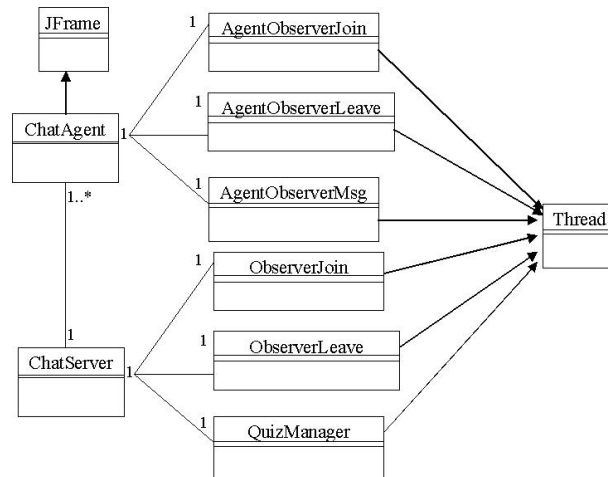


Figura 8: Diagramma delle classi

gestisce il quiz e la comunicazione tra gli utenti della chat. Entrambe le classi delegano a due thread il compito di porsi in attesa dell'evento corrispondente all'entrata o all'uscita di un utente dalla chat. La classe **ChatAgent** inoltre crea un altro thread che si occupa di individuare se sono presenti nuovi messaggi nello spazio di tuple dell'utente ed eventualmente visualizzarli. **QuizManager** è un thread creato dal server che si occupa di generare le domande per il quiz e di aggiornare il punteggio degli utenti. I thread sono utili perchè i compiti vengono distribuiti fra i vari processi anche se la fase di startup risulta appesantita dal tempo e dalle risorse necessarie alla loro creazione.

## 6 Conclusioni e sviluppi futuri

Concludendo il modello a tuple risulta molto utile perchè offre un meccanismo sia di comunicazione sia di sincronizzazione e permette di evitare che lo stato dell'applicazione venga incapsulato nei processi che la compongono. In questo progetto si sono voluti sfruttare questi vantaggi realizzando un'applicazione semplice come la chat che abbia però la caratteristica di tollerare i guasti replicando gli spazi di tuple su due nodi.

Gli sviluppi futuri potrebbero riguardare l'implementazione: a) di un algoritmo che scelga le domande per il quiz in modo random e le inserisca nello spazio di tuple **pool**; b) di un meccanismo che stabilisca la coerenza tra i due nodi in seguito ad un crash.

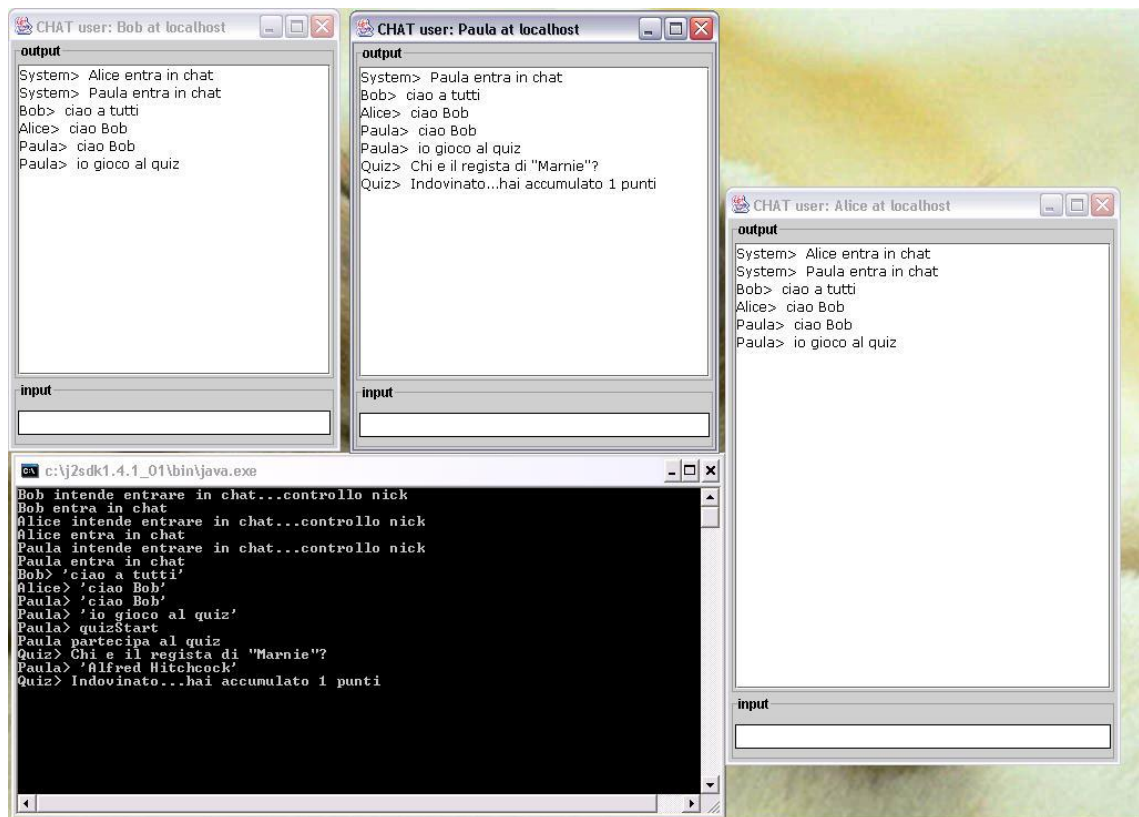


Figura 9: Interfaccia grafica della chat

## Riferimenti bibliografici

- [1] A.Corradi: *Dispense di reti di calcolatori L-S*.  
<http://lia.deis.unibo.it/Courses/RetiLS/>
- [2] A.Ricci: *Tucson guide*.  
<http://lia.deis.unibo.it/research/tucson>
- [3] M.Fowler e K.Scott: *UML Distilled*, Prima edizione italiana. Addison Wesley (2000).