

# Backend

*Anno Accademico 2010-2011*

Federico Viscomi

7 luglio 2011

## Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
<b>2</b>	<b>Global table manager</b>	<b>1</b>
<b>3</b>	<b>Local table manager</b>	<b>3</b>
<b>4</b>	<b>Gtm ed ltm console UI</b>	<b>3</b>
<b>5</b>	<b>Global chat</b>	<b>3</b>
<b>6</b>	<b>Database Manager</b>	<b>4</b>
<b>7</b>	<b>Single table manager e logica del gioco</b>	<b>5</b>
<b>8</b>	<b>Local Bot</b>	<b>6</b>
<b>9</b>	<b>Command line player</b>	<b>6</b>
<b>10</b>	<b>Test del solo backend</b>	<b>6</b>

## 1 Introduzione

Questo documento descrive quali sono state le varie iterazioni che hanno portato all'implementazione del backend e alcuni dettagli di quanto e' stato sviluppato in ogni iterazione.

## 2 Global table manager

In questa iterazione si e' implementato il global table manager o gtm. Il gtm ha diverse responsabilita':

**swarm** : Gestire un insieme(chiamato in seguito swarm) di local table managers o ltm. In particolare gtm deve:

- Permettere di aggiungere e rimuovere ltm dallo swarm.
- Controllare la consistenza dello swarm. A questo scopo fa polling degli ltm chiamando

`LocalTableManagerInterface.isAlive()`

ogni

`GlobalTableManagerInterface.POLLING_DELAY`

millisecondi.

- Inoltrare le richieste delle servlet ad un ltm in modo da bilanciare il carico di lavoro tra gli ltm. A questo scopo gli ltm hanno associata una coppia di interi: il primo indica il numero di tavoli correntemente gestiti dall'ltm e il secondo indica il numero massimo di tavoli gestibili dall'ltm. Quando il gtm deve scegliere un ltm sul quale creare un nuovo tavolo, prende dallo swarm un ltm tra quelli che hanno un carico di lavoro piu' basso. Il carico di lavoro o workload di un ltm e' definito come il rapporto tra i tavoli gestiti e il numero massimo di tavoli gestibili.

Ricapitolando lo swarm deve mantenere associazioni tra interfacce ltm e coppie di interi, deve fornire operazioni di:

- Aggiunta con chiave una interfaccia ltm.
- Rimozione con chiave una interfaccia ltm.
- Scelta di un ltm con workload minimo.

Si e' deciso di usare un ArrayList di record:

`interfaccia ltm, numero di tavoli gestiti, numero di tavoli gestibili`

Tale lista viene mantenuta ordinata con workload crescente. Questa struttura dati minimizza il tempo di esecuzione dell'operazione piu' frequente cioe' la scelta di un ltm. Sia n il numero di ltm allora le complessita' in tempo al caso peggio delle operazioni dello swarm sono:

**aggiunta** :  $O(\log(n))$  in particolare si tratta di una ricerca binaria.

**rimozione**  $O(n)$  perche' in questo caso non vengono confrontati i workload ma le interfacce.

**scelta**  $O(1)$  perche' in questo caso basta scegliere il primo elemento della lista.

**tables** : Mantenere alcune informazioni riguardo tutti i tavoli in Cupido. Tali informazioni sono tutte e sole quelle necessarie ad un giocatore che vuole unirsi ad una partita o che vuole vedere una partita e sono contenute nella classe

`TableInfoForClient`

Ogni tavolo nel gtm e' identificato da un descrittore di tavolo cioe' una istanza di

`TableDescriptor`

Il gtm deve memorizzare associazioni tra descrittori di tavolo e informazioni del tavolo, inoltre deve avere a disposizione le operazioni di:

- Aggiunta usando come chiave un descrittore di tavolo.
- Ricerca usando come chiave un descrittore di tavolo.
- Rimozione usando come chiave un descrittore di tavolo.
- Reperire una parte dei tavoli.

Si e' scelto di memorizzare le informazioni dei tavoli in una tabella hash. L'hash e' calcolato sui campi del descrittore del tavolo. In particolare il descrittore del tavolo contiene: una stringa che identifica un ltm, un intero che identifica il tavolo all'interno dell'ltm. In questo modo le prime tre operazioni richiedono tempo costante. La quarta operazione restituisce una quantita' di tavoli non superiore a

`GlobalTableManagerInterface.MAX_TABLE_LIST_SIZE`

I tavoli da restituire vengono copiati in una istanza di

`ArrayList<TableInfoForClient>`

e vengono scelti in modo casuale. In questo modo la dimensione dei dati inviati al client e' costante. Al caso pessimo la complessita' in tempo di questa operazione e' lineare nel numero di tavoli perche' per scegliere una parte dei valori di una HashMap in modo uniformemente distribuito bisogna eventualmente iterare su tutti essi. Si e' scelto di non fornire una operazione per reperire tutti i tavoli perche' il numero dei tavoli potrebbe essere troppo grande e quindi anche il traffico di rete conseguente. Un'altra strategia presa in considerazione e' quella di un iteratore remoto ma e' stata scartata perche' genera troppe chiamate remote e perche' la lista dei tavoli puo' cambiare tra due chiamate.

Gtm e' usato attraverso RMI quindi il multithreading e' gestito in automatico dalla JVM. La mutua esclusione sul gtm e' gestita semplicemente usando metodi

`synchronized`

E' stata presa in considerazione un'altra strategia di mutua esclusione meno restrittiva cioe': tutte le operazioni hanno accesso esclusivo all gtm con l'eccezione che si permettono piu' operazioni in lettura allo stesso tempo. Tuttavia questa soluzione presenta il problema della starvation delle operazioni di scrittura quindi si e' deciso di non adottarla.

### 3 Local table manager

In questa iterazione si e' implementato il local table manager o ltm. Ltm si occupa di una porzione dei tavoli di Cupido. In particolare ltm fornisce le operazioni seguenti:

- Creare componenti table chiamate in seguito single table manager o stm.
- Rimuovere un stm quando una partita finisce.
- Cercare un stm quando un giocatore vuole unirsi ad una partita o guardare una partita.

Ltm mantiene in una tabella hash gli stm indicizzati da un intero che identifica l'stm in modo univoco nell'ltm. In questo modo le operazioni dell'ltm hanno complessita' in tempo costante. Per quanto riguarda il multithreading e la mutua esclusione valgono le stesse cose dette per il gtm.

### 4 Gtm ed ltm console UI

In questa iterazione sono stati implementati due interpreti di comandi da console minimali per gtm ed ltm. I comandi forniti permettono di vedere lo stato interno del gtm e degli ltm. Quindi questi interpreti hanno una valenza per lo piu' in fase di debug e test. Potrebbero essere usati anche in un deploy reale per operazioni di manutenzione ma si e' scelto di non farlo perche' non serve: in particolare non ci sono requisiti che vietano di riavviare il gtm o gli ltm in fase di esecuzione.

### 5 Global chat

In questa iterazione e' stata implementata la chat globale lato backend. La chat globale e' un buffer di messaggi di dimensione

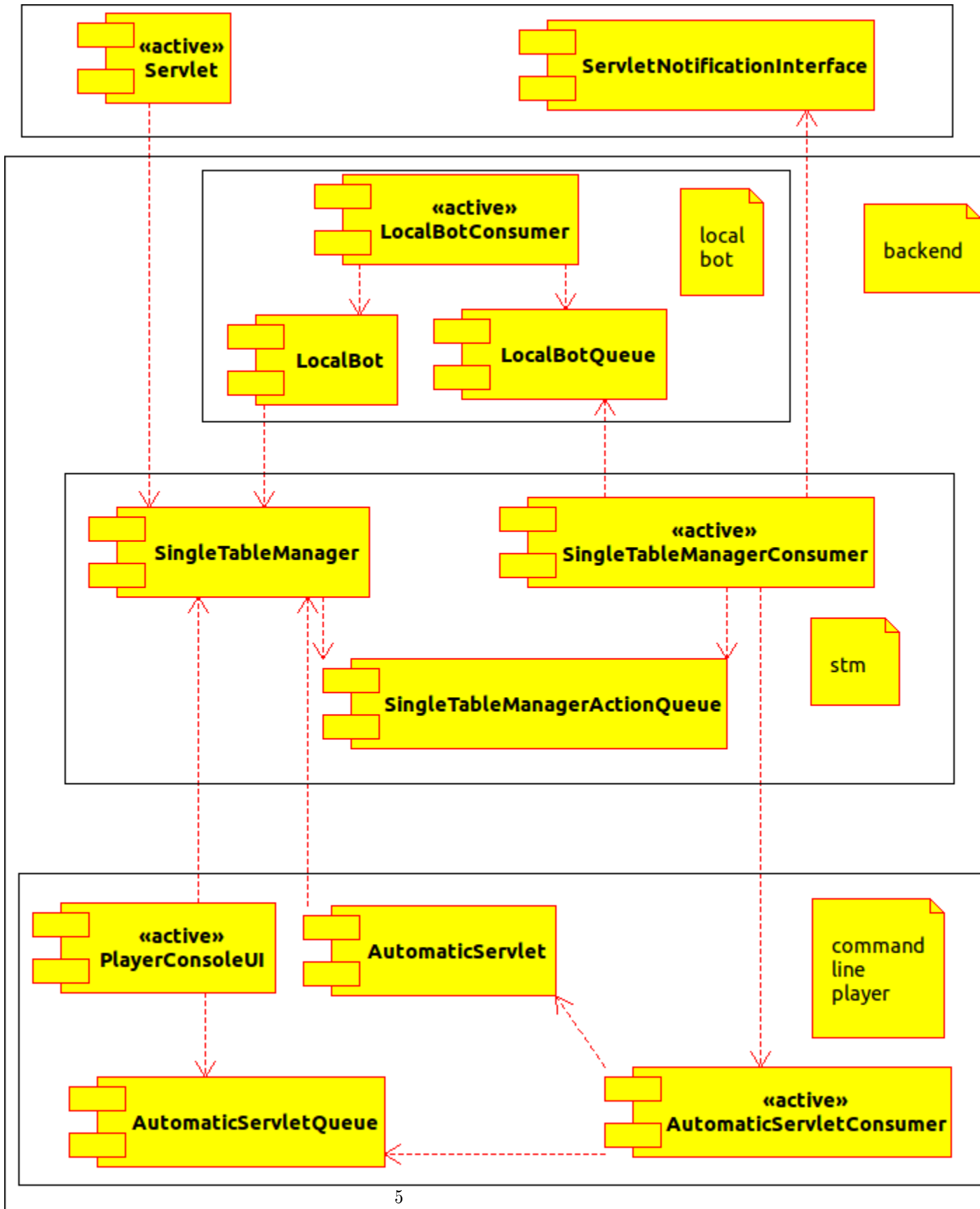
`GlobalChatInterface.MESSAGE_NUMBER`

Gli utenti di Cupido possono solo scrivere messaggi o reperire un certo numero di ultimi messaggi. Quindi i client fanno polling della chat globale e non ricevono notifiche quando un client invia un messaggio nella chat globale. Questo perché si prevede la possibilità di avere un numero elevato di utenti e quindi non conviene che la chat globale memorizzi un'interfaccia di notifica per ognuno di essi e che invii notifiche a questa interfaccia ogni volta che arriva un nuovo messaggio nella chat globale.

## 6 Database Manager

In questa iterazione è stata implementata la classe `DatabaseManager` che fornisce le operazioni descritte in SRS.

## 7 Single table manager e logica del gioco



In questa iterazione e' stato implementato stm. Stm si occupa di una singola partita. In particolare stm:

- gestisce giocatori e bot: permette ad un giocatore di unirsi alla partita di lasciare la partita e di giocare, permette al creatore del tavolo di aggiungere bot, sostituisce un giocatore con un bot se questo lascia la partita dopo l'inizio, notifica i giocatori di quello che succede nella partita.
- gestisce i visitatori: permette ad un visitatore di aggiungersi alla partita o di lasciare la partita, invia ai visitatori le notifiche di quello che succede nella partita.
- gestisce le carte: da le carte e controlla che le mosse dei giocatori siano valide.
- gestisce i punteggi dei giocatori.

Per il multithreading e la mutua esclusione dell'stm valgono le stesse considerazioni fatte per il gtm. Inoltre

## 8 Local Bot

In questa iterazione e' stato implementato un bot, cioe' un giocatore automatico. Quando il bot riceve le carte, le ordina dalla piu' alta alla piu' bassa e poi in base al seme. Quando deve passare le carte sceglie le prime tre che ha. Quando deve giocare una carta sceglie la prima carta valida che ha. L'stm tratta i bot quasi come se fossero delle servlet. Come si vede dalla figura 1 la componente

`LocalBotConsumer`

ha un proprio flusso di esecuzione. Questa

## 9 Command line player

In questa iterazione e' stato implementato un interprete di comandi testuale per un utente di Cupido. L'interprete fornisce una ampia gamma di comandi: creare un tavolo, ottenere la lista di tutti i tavoli, unirsi ad una partita, vedere una partita e giocare una partita. Per una descrizione dei comandi si veda la documentazione della classe

`PlayerConsoleUI`

I comandi possono essere presi da console o da un file di input. Usando un file di input si puo' far giocare una partita a degli utenti in modo completamente automatico.

## 10 Test del solo backend

La cartella `cupidoBackendImpl/test` contiene i file per i test automatici del backend. Sono stati eseguiti i seguenti cinque test automatici sul backend:

- Un giocatore crea un tavolo, aggiunge tre bot e gioca. Questo test si esegue con il comando

`./cupidoBackendImpl/test/test1`

il quale esegue un giocatore a riga di comando che legge i comandi dal file

`./cupidoBackendImpl/test/createTableAndAddThreeBot`

- Un giocatore crea un tavolo e aggiunge due bot, in seguito un altro giocatore si unisce alla partita. Questo test si esegue con il comando

`./cupidoBackendImpl/test/test2`

il quale esegue due giocatori a riga di comando che leggono i comandi dai file:

`./cupidoBackendImpl/test/createTableAndAddTwoBot, ./cupidoBackendImpl/test/joinATable`

- Un giocatore crea un tavolo, aggiunge tre bot e gioca, nel frattempo un visitatore guarda la partita. Questo test si esegue con il comando

`./cupidoBackendImpl/test/test3`

il quale esegue un giocatore come nel primo test e in più esegue un visitatore da riga di comando che legge i comandi dal file

`./cupidoBackendImpl/test/view`

- Un giocatore crea un tavolo, aggiunge tre bot, gioca per un po e poi abbandona la partita. Questo test si esegue con il comando

`./cupidoBackendImpl/test/test4`

il quale esegue un visitatore ed un giocatore da riga di comandi che legge i comandi dal file

`./cupidoBackendImpl/test/createTableAndAddThreeBotLeave`

- Un giocatore crea un tavolo e aggiunge due bot, nel frattempo un altro giocatore si unisce alla partita e in seguito i due giocatori giocano. Dopo qualche turno il giocatore non creatore lascia la partita e viene rimpiazzato da un bot. Questo test si esegue con il comando

`./cupidoBackendImpl/test/test5`

Tutti i test sono andati a buon fine.