Sistema di Gestione Slot Machine

Relazione elaborato di Sistemi Distribuiti

Lo Iacono Luca [1] - Magnani Antonio [2] - Pavllo Andi [3]

1 agosto 2013

[1]mat. 0000651821 - luca.loiacono2@unibo.it
[2]mat. 0000680727 - antonio.magnani@unibo.it
[3]mat. 0000677268 - andi.pavllo2@unibo.it

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione e Obiettivo

L'obiettivo principale della realizzazione del sistema in oggetto è quello di creare un *sistema distributo*. Con sistema distribuito si intende un insieme di entità computazionali concepite come un singolo sistema coerente. Si assume che queste entità siano indipendenti ed autonome, oltre che eterogenee per quanto concerne la loro natura, struttura e comportamento. Sono necessarie dunque:

- *Collaborazione*: le entità devono lavorare come un unico sistema coerente;
- *Amalgamazione*: le entità devono vedersi come un singolo sistema uniforme.

Caratteristiche principale del sistema sarà la trasparenza che permette all'utente di non percepire se le risorse utilizzate sono remote o locali. La trasparenza si evidenzia in diversi contesti:

- *Accesso*: non è necessario mostrare le differenze nella rappresentazione dei dati, e in che maniera si acceda ad una risorsa;
- *Locazione*: non è necessario far conoscere dove sia fisicamente locata una risorsa, e quale sia la posizione dell'utente;

- *Migrazione*: non è necessario mostrare che una risorsa possa muoversi in una locazione differente da quella in cui è al momento;
- *Rilocazione*: non è necessario mostrare che una risorsa possa essere stata spostata da un'altra parte mentre la si stava utilizzando;
- *Replicazione*: non è necessario mostrare che una risorsa sia replicata in un'altra locazione fisica;
- *Concorrenza*: non è necessario mostrare che l'accesso ad una risorsa possa essere avvenuta in maniera competitiva tra diversi utenti;
- *Fallimento*: non è necessario mostrare le volte in cui si è fallito nel tentativo di accedere ad una risorsa, specialmente se il sistema funziona ugualmente.

Il sistema dovrà permettere alle nuove tecnologie e a tecnologie future di poter essere integrate con quanto già implementato precedentemente, è quindi necessario tenere in considerazione l'apertura del sistema rendendolo quindi:

- *Interoperabile*: quanto è difficile far interagire un componente/il sistema stesso con altri, differenti, ma basati sugli stessi standard;
- *Portabile*: quanto del sistema si può spostare e far funzionare su altri sistemi distribuiti;
- *Estensibile*: quanto è difficile aggiungere nuove componenti e funzionalità al sistema.

Considerando che il numero degli utenti potrà essere notevole e la loro distribuzione geografica estesa è necessario progettare un sistema scalabile, quindi capace di gestire grosse quantità di dati e di utenti. Per risolvere questo fenomeno è necessaro distribuire il sistema geograficamente e computazionalmente, facendo attenzione però a mantenere la sicurezza e la continuità dell'elaborazione. Le tecniche principali per sovvenire a questi problemi sono:

- *Nascondere il tempo di latenza della comunicazione*: Tramite comunicazioni asincrone che permettono al richiedente di non fermarsi semplicemente ad attendere la risposta (tuttavia questo non è sempre possibile, e.g. il web.)
- *Distribuire*: Prendendo un componente, dividendolo in parti e distribuendolo lungo tutto il sistema (e.g. DNS.)
- *Replicare*: Copiando determinate risorse in più locazioni per aumentare le performance d'accesso. Questo però introduce un problema di consistenza: i dati replicati devono essere aggiornati in parallelo, e talvolta i tempi di attesa potrebbero non essere tollerabili.

1.1 Specifiche

L'obiettivo di questo elaborato consiste nella realizzazione di un sistema di slot machine distribuito con la predisposizione all'utilizzo di piattaforme di gioco diverse. La società Sisal, fornitrice della piattaforma, vuole permettere a tutti i suoi clienti di utilizzare lo stesso strumento di gioco, distribuito tra le varie organizzazioni che possono comprendere sale slot e siti internet.
Le varie piattaforme di gioco offrono la possibilità di impostare il valore della giocata a 0.50, 1.00 o 2.00 €. Esse prevedono una vincita rappresentata dalla combinazione di tre immagini uguali; a seconda del tipo di immagine, viene calcolato l'importo vinto moltiplicando la somma giocata per un moltiplicatore abbinato. I casi di vincita sono i seguenti:

- 3 Ciliegie: valore giocato x 1;
- 3 Prugne: valore giocato x 2;
- 3 Banane: valore giocato x 3;
- 3 Cocomeri: valore giocato x 5;
- 3 Monete: valore giocato x 10;

- 3 Campane: valore giocato x 10;
- 3 Bar: valore giocato x 50;
- 3 Sette: valore giocato x 100.

Oltre alla vincita tradizionale è possibile vincere un Jackpot che può essere di due tipologie:

- *Jackpot locale*: ottenuto dall'accumulo di un fondo all'interno della singola organizzazione;
- *Jackpot nazionale*: ottenuto dall'accumulo di un fondo composto da tutte le organizzazioni che utilizzano la piattaforma fornita da Sisal.

Se si vince uno dei due Jackpot, o entrambi, non è prevista la vincita tradizionale.

Ogni singola postazione di gioco è un client che regolarmente si interfaccia ad un server locale (che gestisce il Jackpot locale) e ad un server globale (che gestisce il Jackpot nazionale). Per evitare che i Jackpot ripartano da zero a seguito di una vincita, viene regolarmente accantonata una piccola parte delle giocate che vanno a costituire un fondo di ripartenza. Gli incassi di ogni postazione di gioco sono così suddivisi:

- 75%: Vincita diretta garantita (tramite vincita tradizionale);
- 10%: Tassa statale;
- 5%: Jackpot locale:
  - 4%: Jackpot immediato;
  - 1%: Fondo per ripartenza Jackpot;
- 5%: Jackpot nazionale:
  - 4%: Jackpot immediato;

  - 1%: Fondo per ripartenza Jackpot;

- 5%: Compenso fornitore del servizio (Sisal).

Tutti gli utenti registrati al sistema potranno accedere da qualsiasi piattaforma e controllare lo storico dei movimenti effettuati, il proprio saldo ed effettuare prelievi o depositi durante la propria sessione di gioco.
L'elaborato presenta un client di gioco per PC ma il sistema è progettato per permettere ad applicazioni (per smartphone o postazioni installate in sale slot) di integrarsi con esso.

Capitolo 2

Tecnologie utilizzate

2.1 Java Agent Development Framework (JADE)

Java Agent DEvelopment Framework (JADE) è un framework per lo sviluppo di sistemi ed applicazioni distribuite, basate sul paradigma di programmazione ad agenti, rispettanti gli standard FIPA (*Foundation for Intelligent Physical Agents*) relativi ad agenti intelligenti. JADE è un software open source e viene distribuito con licenza LGPL. Include 2 principali componenti:

- Una piattaforma conforme con gli standard FIPA;
- Package per lo sviluppo di agenti in Java.

La Figura 2.1 rappresenta un sistema JADE distribuito su più host conforme alle specifiche.

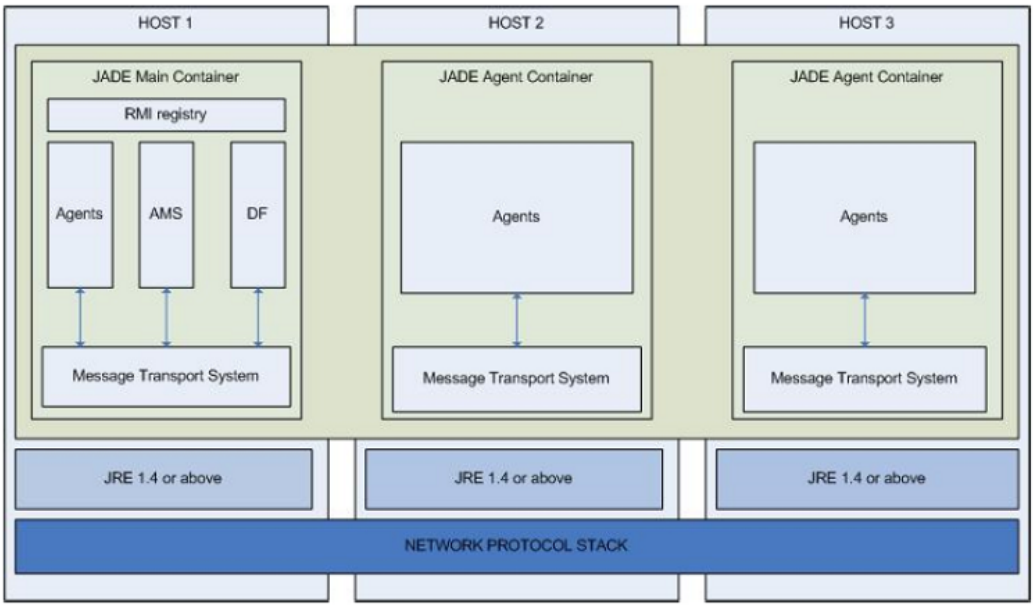

Figura 2.1: Architettura di un sistema distribuito con JADE.

Tabella 2.1: Descrizione componenti di un'architettura JADE

Sottosistema	**Descrizione**
Agent Management System (AMS)	- Controllo accessi da/verso la piattaforma agente; - Uno per ogni piattaforma; - Mantiene la directory degli identificativi agenti (servizio pagine bianche); - Servizio ciclo di vita di ciascun agente; - Ogni agente deve essere registrato sull'AMS per avere un identificativo AID; - Viene avviato dopo il lancio della piattaforma JADE.

Continua nella prossima pagina

Tabella 2.1: continua dalla pagina precedente

Sottosistema	Descrizione
Directory Facilitator (DF)	- Un agente può pubblicizzare i propri servizi e/o ricercarne da altri agenti (servizio pagine gialle); - Viene avviato dopo il lancio della piattaforma JADE.
Message Transport System	- Controlla lo scambio di messaggi interni alla piattaforma; - Controlla lo scambio di messaggia da/verso la piattaforma; - Avviato dopo il lancio della piattaforma JADE.

In accordo con le specifiche FIPA, la comunicazione tra agenti avviene tramite message passing asincrono dove i messaggi, oggetti della classe ACLMessage, sono scambiati. I principi della comunicazione tra agenti, mediante messaggi ACL e code private, sono descritti nella figura seguente: Per

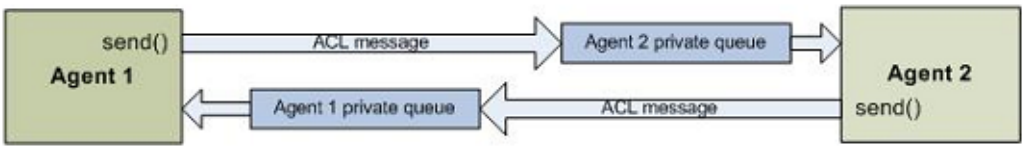

Figura 2.2: Comunicazione tra agenti mediante messaggi ACL.

l'invio di messaggi ad altri agenti viene utilizzato il metodo `send()` della classe `Agent`. I messaggi sono inviati alla coda privata di un particolare agente; la dimensione di tale coda può essere impostata mediante il metodo `setQueueSize()`. Esistono due differenti modalità con cui un agente può ricevere messaggi da altri agenti:

- `Non-blocking`: l'agente non si sospende e continua l'esecuzione dei behaviour (avviene tramite il metodo `receive()`);

- `Blocking`: causa la sospensione di tutte le attività dell'agente, possono essere specificati parametri di timeout in grado di descrivere il tempo massimo di sospensione dell'agente (avviene tramite il metodo `blockingReceive()`).

Nello scambio di messaggi ACL tra agenti un ruolo fondamentale è ricoperto dalle *Ontologie*. Con ontologia si intende una rappresentazione formale, condivisa ed esplicita di una concettualizzazione di un dominio di interesse. Affinché gli agenti di una piattaforma possano comunicare, è necessario che essi condividano non solo linguaggi e protocolli di comunicazione, ma anche un comune vocabolario di termini utilizzati nei messaggi scambiati. La definizione di tale vocabolario in JADE è basata sul concetto di ontologia.

Un agente si compone di un singolo thread di esecuzione e tutti i suoi compiti sono modellati, e possono essere implementati, come oggetti *behaviour*. I behaviour possono essere aggiunti ed eliminati dallo scheduler non-preemptive di un agente tramite i metodi `addBehaviour(Behaviour)` e `removeBehaviour(Behaviour)`. Sono previste tre tipologie di behaviour, realizzate come sottoclassi della classe behaviour: *SimpleBehaviour*, *CompositeBehaviour* e *WrapperBehaviour*. La classe `Behaviour` è una classe astratta e imposta le basi per lo scheduling dei singoli behaviour e le transizioni di stato dell'oggetto.
Tra i metodi più importanti all'interno di questa classe abbiamo:

Tabella 2.2: Descrizione metodi della classe Behaviour.

Metodo	Descrizione
`block()`	- Permette di bloccare il behaviour fino all'avvenire di un determinato evento (fino a quando arriva un messaggio); - Non modifica altri behaviour; - Posiziona il behaviour nella coda dei behaviour bloccati e avrà effetto non appena il metodo `action()` "ritorna"; - Tutti i behaviour bloccati sono rischedulati quando un nuovo messaggio arriva; - Può avere parametri di timeout che consentono blocking temporanei del behaviour a seconda dei millisecondi specificati.
`action()`	- Esegue il behaviour.
`restart()`	- Riavvia esplicitamente il behaviour.
`onStart()`	- Viene eseguito prima dell'esecuzione del metodo action del behaviour.
`onEnd()`	- Chiamato dopo che il behaviour ha completato ed è stato rimosso dal pool dei behaviours dell'agente.
`reset()`	- Ripristina lo stato iniziale del behaviour.

2.2 MySQL - phpMyAdmin

MySQL, definito *Oracle MySQL*, è un *Relational database management system (RDBMS)*, composto da un client con interfaccia a riga di comando e

un server, entrambi disponibili sia per sistemi Unix o Unix-like come GNU/-Linux che per Windows, anche se prevale un suo utilizzo in ambito Unix. La scelta degli strumenti di amministrazione di MySQL è ricaduta su uno dei più importante *MySQL Manager* ovvero *phpMyAdmin*. phpMyAdmin è un'applicazione web scritta in PHP, rilasciata con licenza GPL, che consente di amministrare un database MySQL tramite un qualsiasi browser. L'applicazione è indirizzata sia agli amministratori del database, sia agli utenti. Gestisce i permessi prelevandoli dal database MySQL. phpMyAdmin permette di creare un database da zero, creare le tabelle ed eseguire operazioni di ottimizzazione sulle stesse. Presenta un feedback sulla creazione delle tabelle per evitare eventuali errori. Sono previste delle funzionalità per l'inserimento dei dati (popolazione del database), per le query, per il backup dei dati, ecc.. L'amministratore ha anche a disposizione un'interfaccia grafica per la gestione degli utenti: l'interfaccia permette l'inserimento di un nuovo utente, la modifica della relativa password e la gestione dei permessi che l'utente ha sul database.

2.3 Apache Tomcat

L'architettura da noi realizzata, come verrà descritto in seguito, presenta diversi server raggiungibili in rete. Attualmente tali server sono predisposti in macchine locali ma potrebbero essere installati su una qualsiasi macchina senza dover cambiare configurazione o codice. È necessario trovare un Server in grado di gestire applicazioni web con linguaggio Java, lo stesso con cui è stato sviluppato l'intero ambiente di JADE. *Apache Tomcat* (o semplicemente *Tomcat*) è un contenitore servlet open source sviluppato dalla Apache Software Foundation. Implementa le specifiche JavaServer Pages (JSP) e Servlet di Sun Microsystems, fornendo quindi una piattaforma software per l'esecuzione di applicazioni Web sviluppate in linguaggio Java. La sua distribuzione standard include anche le funzionalità di web server tradizionale, che corrispondono al prodotto Apache. Tomcat è rilasciato sotto la Licen-

za Apache, ed è scritto interamente in Java; può quindi essere eseguito su qualsiasi architettura su cui sia installata una JVM.

Capitolo 3

Progettazione

3.1 Schema di Progetto

Per definire la struttura fisica e logica del sistema è stato creato uno schema di progetto che rappresenta l'architettura del sistema e l'interazione tra gli agenti distribuiti sui diversi server previsti.

Lo schema di progetto presentato in Figura 3.1 identifica l'esistenza di tre server che gestiscono le giocate (i quali possono essere estesi ad un numero superiore nel caso il numero utenti del sistema aumenti considerevolmente), un server dove risiede il database e l'agente che lo gestisce ed infine un server globale che contiene il database, con i dati relativi al Jackpot Nazionale, e l'agente che si occupa di elaborare i calcoli relativi ad esso. Ognuno dei tre server citati contiene un *agente ricevitore* il cui scopo è prendere in carico la giocata inviata da ogni client e, successivamente, smistarla tra i vari *agenti di calcolo* presenti sullo stesso server. L'agente calcolo a sua volta elabora la giocata, comunica con l'*agente Jackpot Nazionale* un'eventuale vincita del Jackpot e informa l'*agente Database* sull'esito della giocata per permettergli di registrarla. Il client attua una comunicazione con l'agente Database per effettuare il login, per richiedere le informazioni sui movimenti e per la gestione dei propri dati.

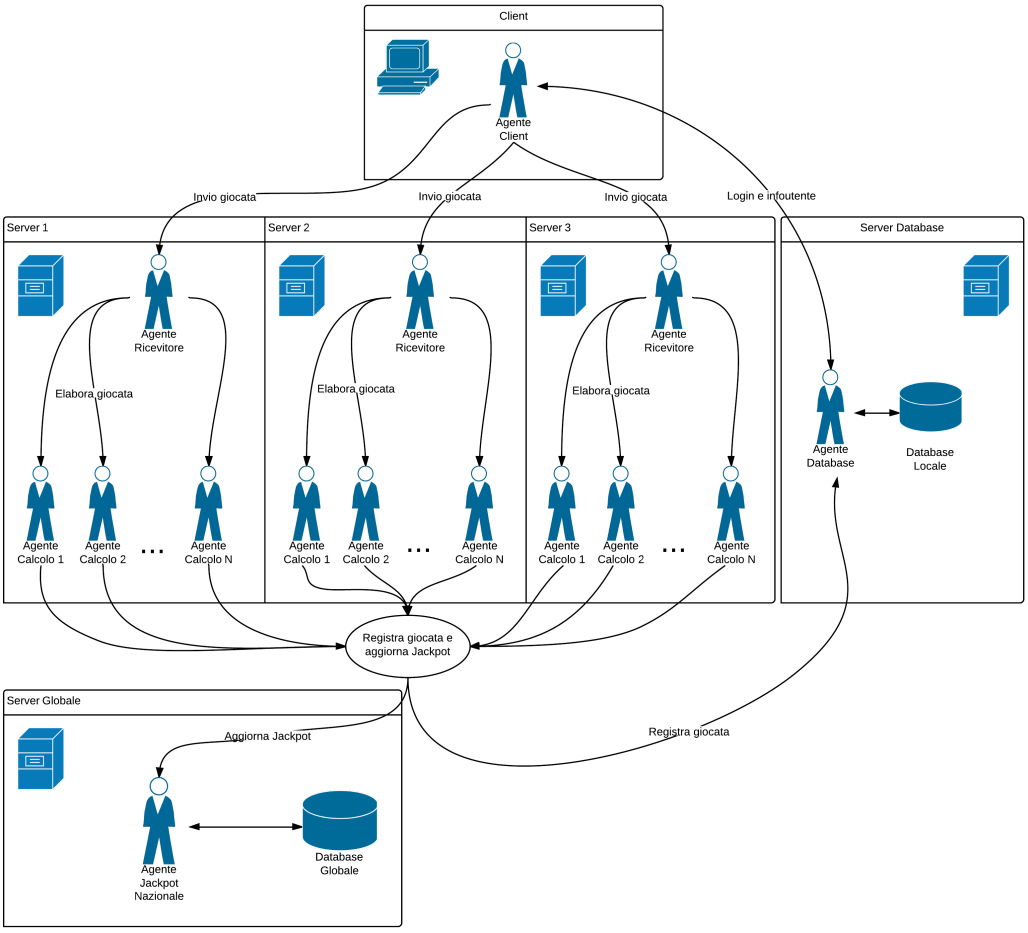

Figura 3.1: Architettura del Sistema Slot Machine.

3.2 Casi d'uso

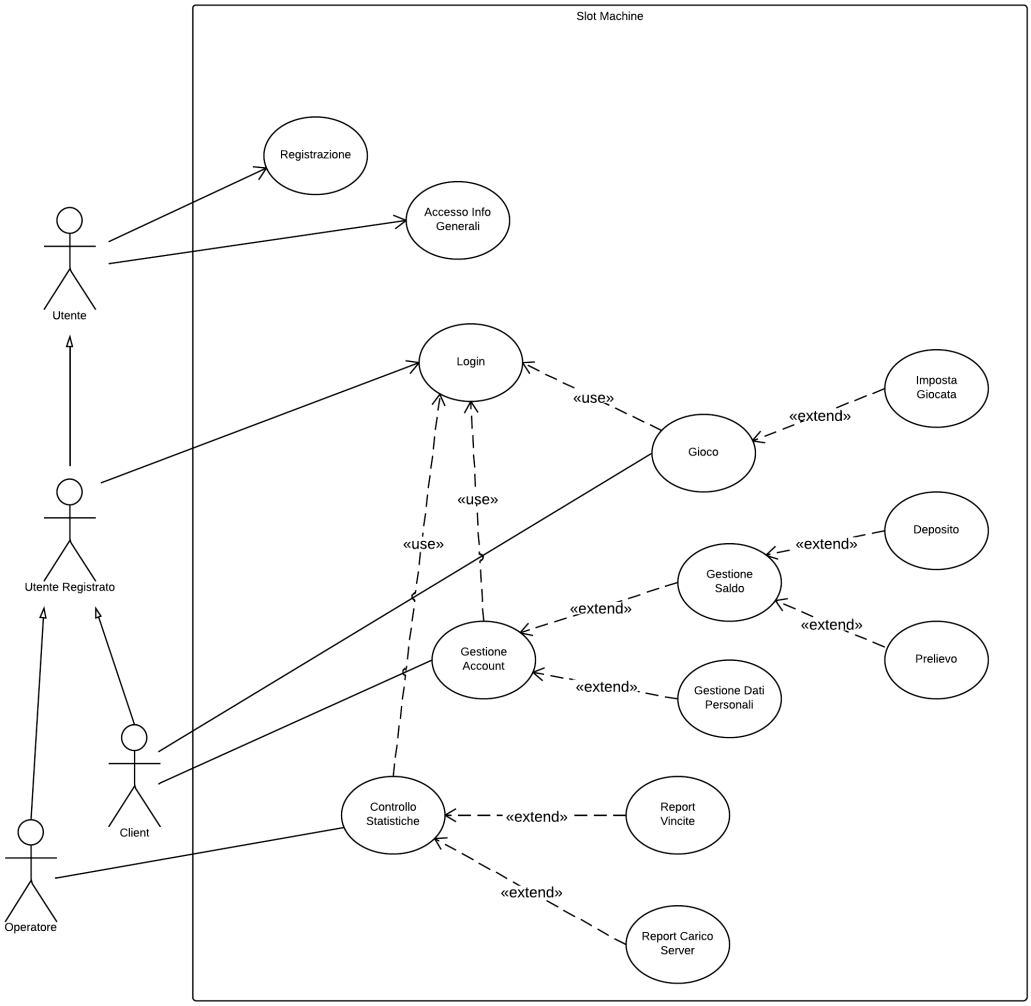

Figura 3.2: Diagramma dei Casi d'Uso.

Analizzando il sistema si individuano 4 distinti attori primari:

- Utente;
- Utente Registrato;
- Client;

- Operatore.

Con *Utente* identifichiamo un attore che non ha ancora credenziali proprie e, pertanto, può interagire con il sistema soltanto tramite i casi d'uso *Registrazione* e *Accesso alle Info Generali*. La procedura di registrazione è necessaria al fine di poter usufruire dei servizi del sistema; il secondo caso d'uso ha il puro scopo di informare l'eventuale cliente delle condizioni d'uso e delle modalità del servizio. L'attore *Utente Registrato* caratterizza quegli utilizzatori che già dispongono di credenziali per accedere al sistema ed è, per l'appunto, la generalizzazione degli attori *Client* e *Operatore*. Un utente registrato esegue il *Login* al fine di poter accedere ai casi d'uso previsti dalla specializzazione di utente registrato a cui appartiene. Nel caso si tratti di un *Client* il caso d'uso prioritario, e fulcro del sistema, è la procedura che consente ad un utilizzatore di fare le proprie giocate. Un *Client* può inoltre usufruire della *Gestione Account* mediante la quale può attuare movimenti sul proprio conto, in entrata e in uscita, e gestire/modificare i propri dati personali. L'attore *Operatore* ha accesso ad un caso d'uso di carattere informativo ovvero il *Controllo Statistiche* mediante il quale può valutare i carichi di lavoro del server o l'ammontare delle vincite.

3.3 Database

Essendo il progetto incentrato sulla realizzazione di un sistema atto ad elaborare e memorizzare informazioni relative ai giocatori e alle giocate, si è resa necessaria l'adozione di una base dati che rispondesse alle nostre esigenze. Dopo un'attenta analisi, abbiamo deciso di utilizzare la piattaforma *MySQL*, abbinata a *phpMyAdmin* per quanto riguarda la fase di amministrazione, il tutto per avere un sistema semplice e, al tempo stesso, sempre disponibile.

Una volta determinata la piattaforma, è iniziata la fase di progettazione del database. Si è dovuto tener conto di tutti i tipi di operazioni che possono essere svolte dagli utenti e si è cercato di capire in che modo potessero

relazionarsi con i vari jackpot. Lo schema ER ricavato è descritto in Figura 3.3.

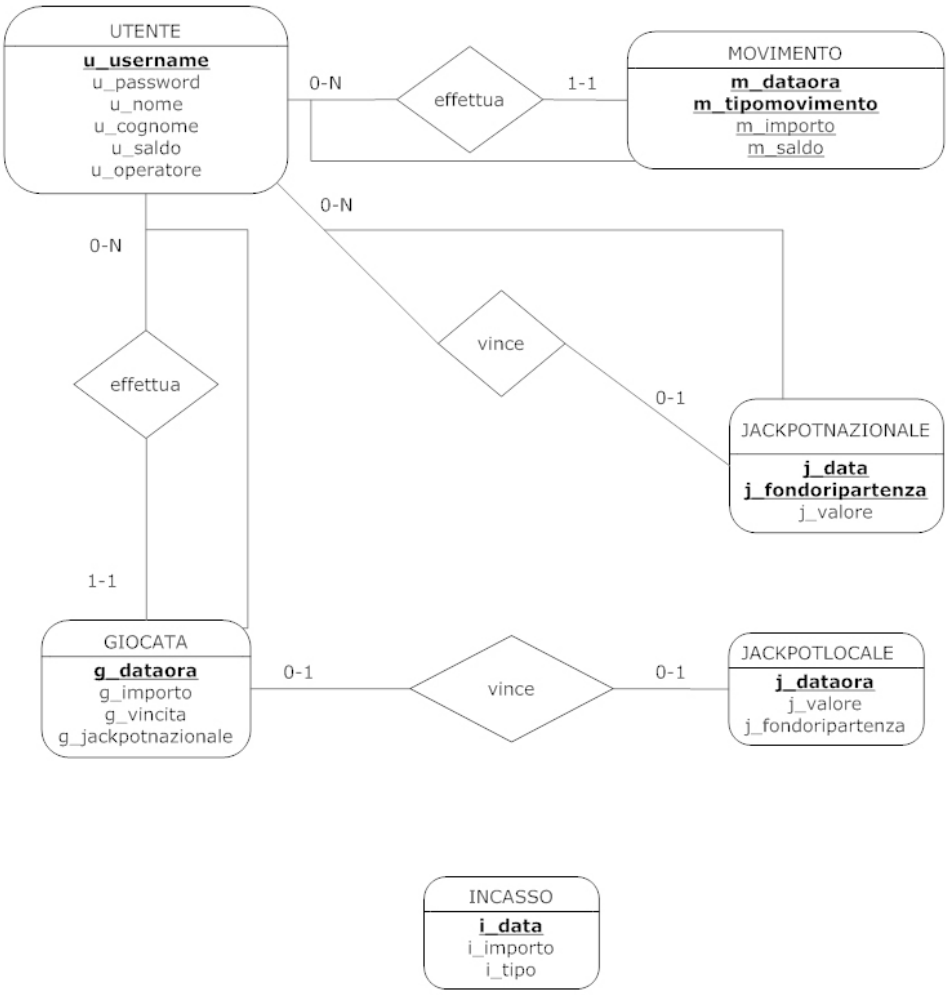

Figura 3.3: Schema ER Slot Machine.

3.3.1 Utente

Per ogni utente vengono memorizzati solamente i dati anagrafici principali (*nome, cognome*), i dati relativi all'account (*username* e *password*, quest'ultima opportunamente criptata in *MD5*), il saldo corrente (viene modificato

ad ogni giocata e movimento) e, infine, un *flag* per identificare se l'utente è un operatore oppure no. Ogni utente può effettuare due tipi di operazioni:

- *Giocata*: l'utente gioca una certa somma nella speranza di vincere;
- *Movimento*: l'utente decide se depositare o prelevare denaro dal proprio conto.

3.3.2 Giocata

Tutte le informazioni relative alla giocata vengono memorizzate all'interno di questa tabella. Esse riguardano la data e l'ora in cui è stata effettuata la giocata, l'utente che l'ha effettuata, l'importo giocato e quello vinto, un *flag* per memorizzare l'eventuale vincita del jackpot nazionale ed, infine, un campo che memorizza la data e l'orario di un'eventuale vincita del jackpot locale.

3.3.3 Movimento

Analogamente alla tabella *Giocate*, la tabella *Movimento* memorizza tutti i movimenti di denaro effettuati all'interno del sistema. Per ogni record vengono memorizzate le informazioni relative alla data e all'orario in cui il movimento è stato effettuato, l'*username* del soggetto, il *tipo di movimento* (*prelievo, deposito, giocata, vincita*) ed il *saldo totale* dell'utente in quel particolare momento.

3.3.4 Jackpot Locale

La tabella relativa alla memorizzazione del Jackpot Locale viene aggiornata ad ogni giocata, inserendo un nuovo record che comprende la data e l'ora dell'aggiornamento ed i nuovi valori relativi al saldo ed al fondo di ripartenza.

3.3.5 Jackpot Nazionale

Al contrario del Jackpot Locale, quello Nazionale viene aggiornato massimo due volte al giorno, cioè a mezzanotte e, in caso di vincita, subito dopo l'accertamento di quest'ultima. Si evince quindi che il Jackpot Nazionale può essere vinto al massimo una volta al giorno. Le informazioni memorizzate riguardano il *giorno di validità* del jackpot, il suo *valore di saldo* e di *fondo ripartenza* e, in caso di vincita, l'*username* del vincitore.

3.3.6 Incasso

Gli incassi vengono effettuati una volta al mese e, per ognuno di essi, vengono memorizzate la *data*, l'*importo* ed il *tipo di incasso* (*sisal, guadagno, tasse* ecc.) in base alle percentuali indicate in precedenza.

3.4 Diagrammi di Sequenza

3.4.1 Giocata

La giocata è l'attività centrale del sistema, proprio per questo motivo il numero di entità che prenderanno parte a questo processo è molto elevata. Esse si dividono in *client*, *database* e vari agenti, ognuno con uno scopo specifico. La giocata parte ovviamente dal client e contiene tutte le informazioni riguardanti l'utente e l'importo della giocata. Essa viene inviata al `RicevitoreAgent`, il cui compito è quello di instradarla verso un agente che sia in grado di elaborare la giocata e che sia libero in quel particolare istante.

Il `RicevitoreAgent` controlla lo stato di tutti gli `ElabGiocataAgent` sotto la sua supervisione, appena ne trova uno libero gli inoltra il messaggio relativo alla giocata e lo imposta come occupato. Nel caso non vi fossero `ElabGiocataAgent` liberi, il messaggio verrà inviato ad un altro `RicevitoreAgent`, il quale supervisiona un altro insieme di `ElabGiocataAgent`.

Superata questa fase, il messaggio arriva all'`ElabGiocataAgent`, il quale ha il compito di gestire tutte le fasi successive della giocata, fino ad invia-

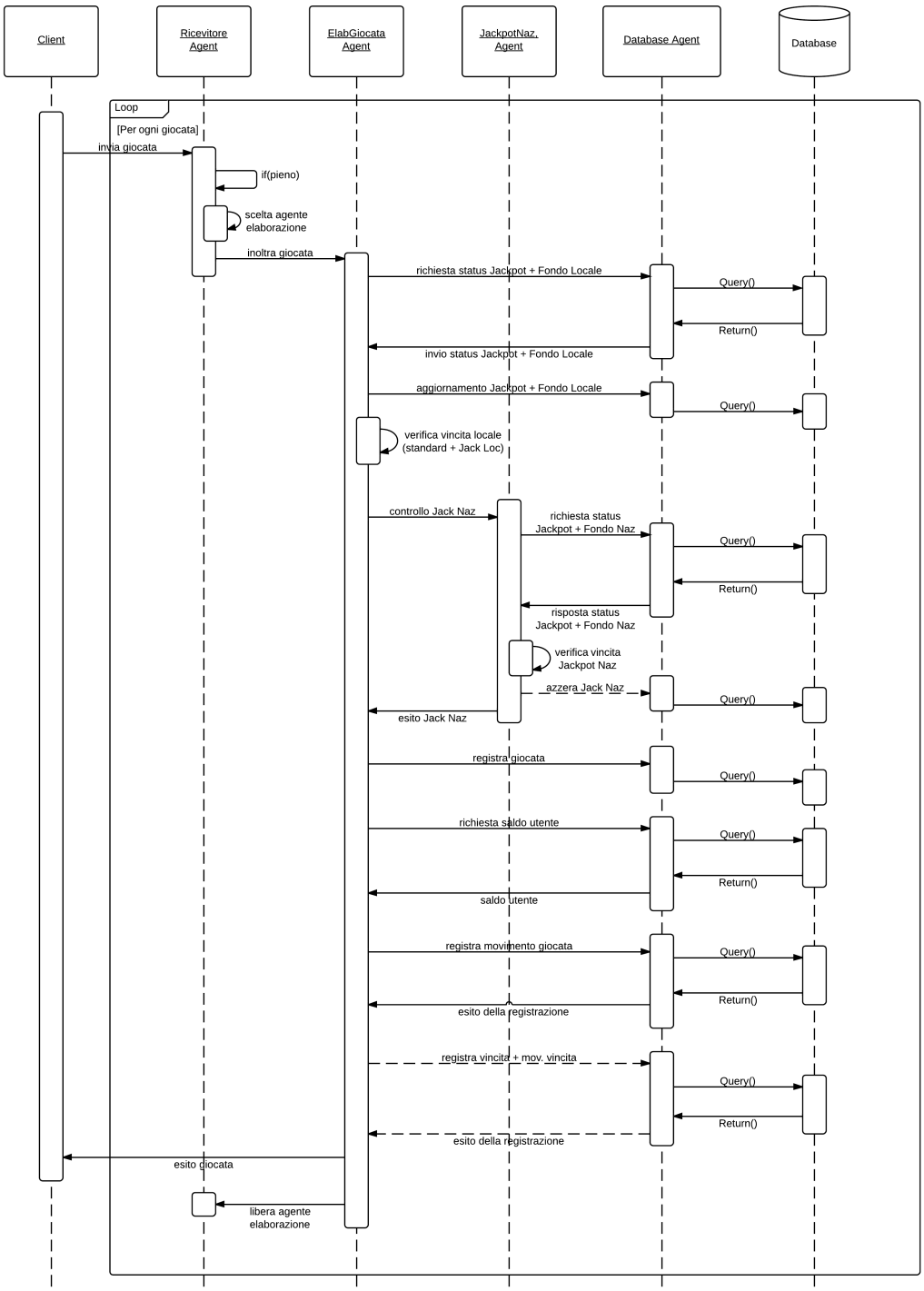

Figura 3.4: Diagramma di sequenza della giocata.

re l'esito di essa al client. L'agente come prima cosa si interfaccia con il `DatabaseAgent` per richiedere i valori relativi al saldo ed al fondo di ripartenza del Jackpot Locale. Il `DatabaseAgent` ha come unico scopo quello di interfacciarsi con la base di dati a scopo di inserimento, aggiornamento e reperimento di informazioni, il tutto attraverso l'utilizzo di query SQL. Una volta ricevute le informazioni relative al Jackpot Locale, l' `ElabGiocataAgent` è in grado di calcolare il nuovo Jackpot e di inviare la richiesta di aggiornamento al `DatabaseAgent`.

La fase successiva è probabilmente quella più importante, l'agente infatti deve verificare l'eventuale vincita della giocata (jackpot oppure vincita standard). In caso di vincita del Jackpot Locale, l'agente dovrà inviare una richiesta di azzeramento del Jackpot Locale. Una volta controllata la vincita locale verrà inviato il messaggio contenente la giocata al `JackpotNazionaleAgent`. Esso contatta immediatamente il `DatabaseAgent` per avere informazioni relative al saldo ed al fondo di ripartenza del Jackpot Nazionale. La fase seguente consiste nella verifica dell'eventuale vincita del Jackpot Nazionale: in caso di vincita si procede ad inviare una richiesta di azzeramento del Jackpot. Per concludere, il `JackpotNazionaleAgent` invia all'`ElabGiocataAgent` un messaggio contenente la giocata aggiornata.

Arrivati a questo punto la gestione della giocata torna nuovamente in mano all'`ElabGiocataAgent`. Vengono inviate al `DatabaseAgent` le informazioni relative alla memorizzazione della giocata e una richiesta del saldo relativo all'utente che la sta effettuando. Una volta ricevuta quest'informazione, viene inviata una richiesta di scrittura relativa al movimento della giocata, in caso di vincita bisognerà richiedere un'ulteriore scrittura del movimento relativa a quest'ultima. Una volta arrivata la conferma positiva da parte del `DatabaseAgent`, l'`ElabGiocataAgent` comunica l'esito della giocata al client e, successivamente, invia un messaggio al `RicevitoreAgent` con lo scopo di liberare l'`ElabGiocataAgent` che ha gestito la giocata. Il `RicevitoreAgent` riceve la richiesta e procede a liberare il mittente.

3.4.2 Agente Demone

L'agente demone si occupa di aggiornare i valori relativi all'incasso giornaliero dell'intero sistema. Ogni giorno a mezzanotte vengono estratti il totale delle giocate effettuate e il totale delle vincite, questo al fine di calcolare le diverse percentuali di distribuzione dell'incasso (jackpot nazionale, fondo jackpot nazionale, sisal e tasse) e il guadagno del gestore del sistema.

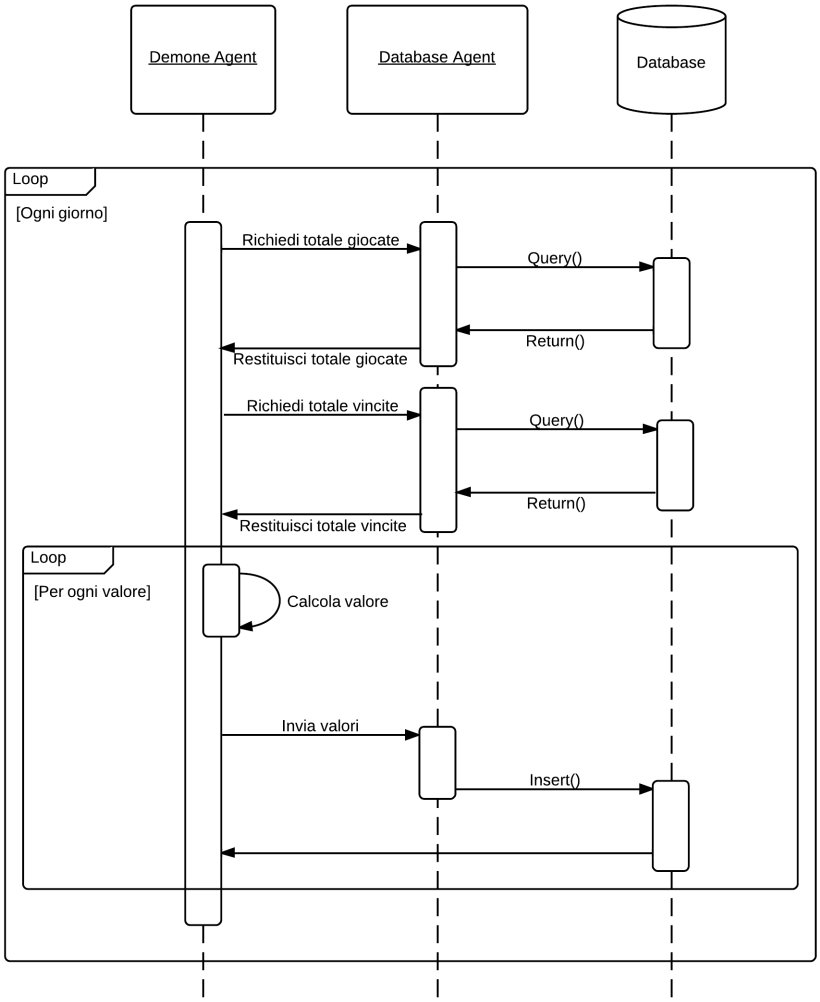

Figura 3.5: Diagramma di sequenza dell'agente Demone.

Capitolo 4

Implementazione

4.1 Server-side

Il server-side si può dividere in due componenti:

- Un'infrastruttura atta a gestire l'instradamento, l'elaborazione delle giocate e la gestione del jackpot locale;

- Un'infrastruttura il cui scopo è quello di verificare la vincita del jackpot nazionale.

La prima componente sarà interamente controllata dal fornitore del servizio. Ogni fornitore potrà possedere uno o più server in cui mantenere gli agenti necessari alla giocata locale. Questi server potranno essere replicati per far fronte a guasti e congestioni, inoltre dovrebbero essere localizzati in aree geografiche differenti a scopo preventivo.

La seconda infrastruttura sarà controllata da un ente nazionale. Ad essa si interfacceranno tutti i fornitori del servizio per verificare la vincita del jackpot nazionale. Anche in questo caso valgono le stesse osservazioni fatte in precedenza sul replicamento.

4.1.1 RicevitoreAgent

Lo scopo di questo agente è principalmente quello di instradare le giocate, provenienti dai vari client, verso gli `ElabGiocataAgent` controllati da `RicevitoreAgent`. L'agente è composto da due *Behaviour*, il primo è un *OneShotBehaviour* all'interno del quale verrà inizializzata una matrice di dimensione *Nx2*, dove *N* corrisponde al numero degli `ElabGiocataAgent` controllati da `RicevitoreAgent`. La prima colonna conterrà i nomi degli agenti mentre la seconda il loro stato (0 se *libero*, 1 se *occupato*).

Una volta fatto ciò si entra all'interno del secondo *Behaviour*, di tipo *CyclicBehaviour*, che conterrà tutte le funzionalità dell'agente. Inizialmente l'agente resta in attesa di un messaggio ACL, una volta arrivato ne analizza l'ontologia e lo gestisce in funzione di essa. Sono presenti le seguenti *ontology*:

- *richiestaGiocata*: l'agente sa che il messaggio contiene una giocata, quindi il suo compito è quello di instradarla verso un `ElabGiocataAgent`. Per fare ciò controlla la matrice creata precedentemente. A questo punto si hanno due possibili scenari:
  - Viene trovato un `ElabGiocataAgent` **libero**: la giocata viene inviata a lui e viene registrato come *occupato*. Una volta fatto ciò il `RicevitoreAgent` torna in attesa di un altro messaggio.
  - Gli `ElabGiocataAgent` sono **occupati**: l'agente deve inviare la giocata ad un altro `RicevitoreAgent` (il cui indirizzo è memorizzato, insieme a quello di altri agenti dello stesso tipo, in una struttura all'interno dell'agente stesso) che dispone di agenti `ElabGiocataAgent` liberi. Il `RicevitoreAgent` torna in stato di attesa.
- *liberaAgente*: in questo il messaggio proviene da uno degli `ElabGiocataAgent` controllati, il quale, una volta finita la sua esecuzione, richiede di essere liberato. Il `RicevitoreAgent` estrae il numero dell'agente dal messaggio, ne ricerca il valore all'interno della relativa tabella e procede a liberarlo, una volta fatto ciò torna in stato d'attesa.

4.1.2 ElabGiocataAgent

Lo scopo di questo agente è quello di gestire la giocata a livello locale. L'agente accetta solo messaggi aventi *performative* `REQUEST` e *ontology* `richiestaGiocata`. Una volta accertato che il messaggio è del tipo richiesto ne estrae il contenuto, ovvero un oggetto`InfoGiocata` (definito dalla classe `entity.InfoGiocata.java`). L'oggetto contiene solamente i tre valori inizializzati dal client, ovvero *username*, *AIDClient* e *importo (giocato)*. L'agente registra all'interno dell'oggetto il relativo orario e genera un intero random compreso tra `0` e `9999`. Questo valore è molto importante in quanto, nel caso venga estratto 9999, l'utente vincerebbe il Jackpot Locale (cioè 1 possibilità su 10000). Successivamente l'`ElabGiocataAgent` richiede al `DatabaseAgent` i valori relativi al jackpot e al fondo di ripartenza locale, restando bloccato finchè non riceve entrambe le informazioni.

Il passo successivo consiste nell'aggiornare sia il jackpot che il fondo locale, estraendo dall'importo giocato una percentuale (come descritto al Capitolo 1), che saranno nuovamente inviati al `DatabaseAgent` per effettuare l'aggiornamento. A questo punto viene effettuata la verifica della vincita locale, che può essere di due tipi:

- *Vincita Jackpot Locale*: è stato generato random il numero 9999, quindi l'utente ha vinto il Jackpot. Nell'oggetto `giocata` viene inizializzato a *true* l'attributo relativo alla vincita del jackpot locale. A questo punto viene inviata una richiesta di azzeramento del jackpot, ponendo il fondo di ripartenza come nuovo jackpot ed azzerando il nuovo fondo ripartenza. Il tempo dell'aggiornamento sarà posto di default a `dataOraGiocata` + 1 secondo, il tutto per evitare chiavi duplicate.

- *Vincita Giocata Standard*: viene richiamato il metodo `calcola()` all'interno della classe `utility.CalcolaGiocata`. Essa ha lo scopo di generare un random compreso tra 000 e 999; a seconda della combinazione uscita l'utente ha la possibilità di moltiplicare l'importo giocato per il valore corrispondente alla combinazione. In totale vi sono:

- 1 caso di vincita per 100x
- 2 casi di vincita per 50x
- 5 casi di vincita per 20x
- 8 casi di vincita per 10x
- 10 casi di vincita per 5x
- 20 casi di vincita per 3x
- 85 casi di vincita per 2x
- 90 casi di vincita per 1x

Il tutto non è casuale, ma bensì è stato progettato in modo che la Slot Machine paghi, sulla base di 1000 partite, il 75% dell'importo giocato. Nel caso non uscisse nessuna combinazione relativa a questi casi di vincita, il moltiplicatore è da considerarsi 0x, quindi l'utente avrà perso l'importo giocato.

Ad ogni moltiplicatore sopra descritto è associata una combinazione di 3 immagini, come descritto al Capitolo 1. La fase successiva riguarda il controllo del jackpot nazionale.
L' `ElabGiocataAgent` creerà un nuovo messaggio ACL contenente un oggetto di tipo `InfoGiocata`, aggiornato con tutti i campi relativi alla giocata locale, e lo invierà al `JackpotNazionaleAgent`, il quale ha il compito di verificare un'eventuale vincita del jackpot nazionale e di restituire all' `ElabGiocataAgent` l'oggetto, aggiornato negli attributi relativi alla giocata nazionale.

Arrivati a questo punto la parte relativa alla verifica delle vincite è conclusa, è necessario quindi inserire all'interno del database i nuovi record relativi alla giocata e ai movimenti effettuati. Innanzitutto viene inserito il record relativo alla giocata, seguito da una richiesta del saldo totale dell'utente allo scopo di registrare il/i movimento/i. Successivamente viene registrato il movimento relativo alla giocata e il saldo utente viene posto come *nuovoSaldo* = *vecchioSaldo* - *importoGiocato*, ciò avviene anche in caso di vincita. In caso

di vincita verrà inserito un nuovo movimento, in cui il saldo utente viene posto come *nuovoSaldo = nuovoSaldo + importoVinto*.

Una volta che tutti i campi all'interno del database sono stati aggiornati e che la giocata è conclusa, viene inviato al client un messaggio ACL contenente l'oggetto di tipo `InfoGiocata`. Successivamente viene inviata al `RicevitoreAgent` la richiesta di liberare l'agente `ElabGiocataAgent` che ha gestito la giocata.

4.1.3 JackpotNazionaleAgent

Lo scopo dell'agente è quello di verificare un'eventuale vincita del jackpot nazionale e di aggiornare la struttura dati relativa alla giocata in base all'esito della vincita. Il `JackpotNazionaleAgent`, diversamente dagli agenti mostrati in precedenza, è unico: ciò significa che tutti gli `ElabGiocataAgent` si interfacceranno con esso. Anche in questo caso troviamo un *CyclicBehaviour*, all'interno del quale l'agente resta in attesa di un messaggio avente *Performative* `REQUEST` e *ontology* `jackpotNazionale`. Una volta accertatosi che il messaggio ricevuto è del tipo richiesto, l'agente genera un intero random compreso tra 0 e 99999. Questo valore è molto importante perchè, nel caso fosse estratto 99999, l'utente avrebbe vinto il Jackpot Nazionale (cioè 1 possibilità su 100000).

Successivamente l'agente contatta il `DatabaseAgent` per avere informazioni riguardanti il jackpot ed il fondo di ripartenza nazionale. La fase seguente consiste nel verificare che durante la giornata in cui è stata effettuata la giocata il jackpot nazionale non sia stato già vinto, in base a questa verifica si aprono due scenari:

- *Jackpot Nazionale già vinto*: si verifica controllando il valore del fondo di ripartenza nazionale, se esso è pari a 0 significa che il jackpot è stato già vinto. In questo caso si salta tutta la fase di controllo della vincita, anche nell'eventualità venga estratta la combinazione 99999.

- *Jackpot Nazionale non vinto*: in questo caso si controlla la combinazione estratta, se essa è pari a 99999 allora vuol dire che il jackpot è stato vinto. Di conseguenza viene aggiornato l'oggetto `giocata` impostando i valori relativi alla vincita del jackpot nazionale e viene creato un messaggio ACL per aggiornare il jackpot nazionale all'interno del database. Il nuovo jackpot verrà impostato con valore pari al fondo di ripartenza, mentre il nuovo fondo sarà impostato a 0. Inoltre, verrà registrato l'*username* del vincitore del jackpot nazionale.

Ora che il controllo della vincita del jackpot nazionale è concluso, viene comunicato all'agente `ElabGiocataAgent` l'oggetto `giocata` contenente gli attributi relativi al jackpot nazionale aggiornati.

4.1.4 DemoneAgent

L'agente `DemoneAgent` al momento della sua chiamata, cioè all'avvio del sistema, esegue un *Waker Behaviour*(specializzazione di *OneShot Behaviour*) che ha la particolarità di eseguire il suo contenuto dopo una quantità di tempo specificata. Dopo aver calcolato la differenza in millisecondi tra l'istante di avvio dell'agente e la mezzanotte del giorno corrente, è possibile creare il *Waker Behaviour* impostando la sua esecuzione non appena scatta la mezzanotte.

L'esecuzione del *Waker Behaviour* consiste nella chiamata al metodo `aggiornaValori()` e nell'avvio di un *Ticker Behaviour* (specializzazione di *Cyclic Behaviour*) il quale viene eseguito ciclicamente ogni 24 ore e che a sua volta richiama il metodo `aggiornaValori()`.

Il metodo `aggiornaValori()` si occupa di interrogare il `DatabaseAgent` richiedendo il totale delle giocate e delle vincite del giorno corrente. Sulle giocate calcola gli importi riguardanti gli incassi che saranno così suddivisi:

- Tasse: 10% delle giocate;
- Sisal: 5% delle giocate;

- Jackpot nazionale: 4% delle giocate;
- Fondo Jackpot nazionale: 1% delle giocate;
- Guadagno: totale delle giocate al netto delle percentuali sottratte e delle vincite.

Calcolati gli importi essi vengono inviati al `DatabaseAgent` per aggiornare i valori della base di dati. Per gestire al meglio le date e calcolare l'inizio e la fine della giornata viene utilizzata la classe `utility.GestioneDate.java`.

4.1.5 DatabaseAgent

L'agente `DatabaseAgent`, nel momento in cui viene invocato, inizializza la connessione con il database e rimane in ascolto, attraverso un *CyclicBehaviour*, di tutti i messagi con *performative* `QUERY-IF`. In base all'ontologia ricevuta elabora il contenuto del messaggio e formula la rispettiva query al database.

Come detto sopra ad ogni ontologia l'agente provvede a preparare una query da sottoporre al database, in base all'ontologia si sa già quanti parametri prevede la query, nel caso di più parametri da inserire il contenuto del messaggio sarà composto da valori suddivisi dal carattere ; in modo da applicare il metodo `split()` e memorizzare i singoli parametri in un array di `String`. Le ontologie utilizzate sono le seguenti:

- *login*: prevede un messaggio con due parametri (*username* e *password*) le quali vengono sottoposte al database per verificare l'esattezza della combinazione. In risposta al messaggio viene inviata una stringa contenente *true* o *false* a seconda dell'esito del login;
- *infoutente*: riceve l'*username* di un utente all'interno del messaggio e invia due messaggi in risposta, uno contiene l'elenco degli ultimi 20 movimenti dell'utente e il secondo tutte le informazioni dell'utente (*username, nome, cognome e saldo*) concatenate tramite il carattere ;;

- *saldoutente*: riceve l'*username* di un utente all'interno del messaggio e invia in risposta un messaggio contenente il valore del saldo dell'utente;
- *registragiocata*: riceve un oggetto di tipo Giocata che contiene tutte le informazioni da salvare nel database per registrare la giocata;
- *readjacknaz*: prevede l'invio del valore del Jackpot Nazionale in risposta al messaggio;
- *readfondonaz*: prevede l'invio del valore del fondo di ripartenza del Jackpot Nazionale in risposta al messaggio;
- *readjackpotlocale*: prevede l'invio del valore del Jackpot Locale in risposta al messaggio;
- *readfondoripartenzalocale*: prevede l'invio del valore del fondo di ripartenza del Jackpot Nazionale in risposta al messaggio;
- *aggiornavincitorejackpotnazionale*: riceve in input l'*username* del vincitore del Jackpot Nazionale da aggiornare all'interno del database;
- *aggiornajackpotnazionale*: riceve in input il valore del Jackpot Nazionale da aggiornare all'interno del database;
- *aggiornajackpotlocale*: riceve in input il valore del Jackpot Locale da aggiornare all'interno del database;
- *inseriscimovimento*: riceve un oggetto di tipo movimento che contiene tutte le informazioni necessarie per registrare il movimento nel database, restituisce un messaggio di conferma di avvenuta registrazione del record.

Per la gestione dei dati da sottoporre alle query e risultanti da esse sono state create, all'interno del package `Entity`, classi apposite per gestire e memorizzare i dati. Le classi utilizzare per la gestione dei dati sono:

- `InfoGiocata.java`;

- `Utente.java`;
- `Movimento.java`.

L'esecuzione vera e propria delle query al database avviene attraverso le classi contenute nel package `Query` le quali contengono le interrogazioni e sono suddivise secondo l'organizzazione delle tabelle nel database:

- `QueryGiocata.java`;
- `QueryIncasso.java`;
- `QueryJackpot.java`;
- `QueryMovimenti.java`;
- `QueryUtente.java`.

Le password, all'interno del database, vengono salvate in forma criptata attraverso l'algoritmo MD5, viene quindi utilizzata la classe `utility.Codifica.java` per elaborare la password inserita in fase di login. Il metodo `md5(String message)` prende in input la password inserita e restituisce una variabile di tipo String di 32 caratteri contenente la password criptata.

4.2 Client-side

Considerando la tipologia di applicazione e la necessità di offrire un'adeguata interfaccia grafica all'utente, è stato di fondamentale importanza l'utilizzo della classe `jade.gui.GuiAgent`, specializzazione della classica `jade.core.Agent`. In Java una GUI viene eseguita su un opportuno thread che permette di gestire e reagire repentinamente agli eventi generati dell'interazione dell'utente con il sistema, d'altro canto un agente viene eseguito su un proprio thread che permette la gestione dei suoi behaviour. Siccome non è efficiente permettere ad un thread di invocare direttamente i metodi

dell'altro thread, JADE predispone un meccanismo di interazione tra i due thread che consente l'integrazione tra un agente e una GUI. Tale meccanismo si basa semplicemente sul passaggio di eventi (*event passing*) e funziona considerando in entrambe le direzioni di interazione (da GUI ad agente e viceversa).

Tutti gli agenti *client-side* specializzano quindi la classe `GuiAgent` mentre il package `GUI` contiene la definizione di tutte le classi descriventi la GUI associate ai singoli agenti; tali classi implementano l'interfaccia `ActionListener` necessaria alla comunicazione tra agenti e GUI.

4.2.1 LoginAgent

Il primo agente, e quindi la prima interfaccia grafica con cui entra in contatto l'utente, è la schermata di Login. La procedura di Login è ovviamente necessaria per poter usufruire dei servizi offerti quali, ad esempio, mm la possibilità di effettuare giocate e gestire il proprio account. L'interfaccia grafica (`GUI.Login.java`) è molto semplice (Figura 4.1) e si compone di un `JTextField` relativo all'*username* e di un `JPasswordField` relativo alla *password* dell'utente. L'unico evento previsto da questa interfaccia è definito dalla pressione del `JButton` *OK*: in caso ciò avvenga il `LoginAgent` innanzitutto controlla la composizione dei campi relativi ad username e password e successivamente, tramite il metodo `doSend()` interroga il `DatabaseAgent` inviandogli un messaggio ACL, con ontologia *login*, contenente le credenziali digitate dall'utente. L'unico behaviour presente nel `LoginAgent` è un *CyclicBehaviour* che attende una risposta da parte del `DatabaseAgent`: nel caso essa sia affermativa procede alla generazione di un agente `HomeAgent` e, tramite il metodo `DoDelete()`, procede alla terminazione dell'agente di login. Nel caso di risposta negativa da parte del `DatabaseAgente` viene comunicato all'utente l'errore in fase di inserimento delle credenziali.

Figura 4.1: Client: Interfaccia di Login.

4.2.2 HomeAgent

Il menù principale del client è costituito dall'interfaccia definita dal file `GUI.HomePanel.java` a cui è associato l'agente `HomeAgent.java`. Come precedentemente descritto tale agente è figlio diretto del `LoginAgent`. In Figura 4.2 è rappresentata la GUI descritta da `HomePanel.java`. Gli eventi associati a questa interfaccia riguardano la pressione dei `JButton btnPlay, btnLogout, btnAccountInfo`. La pressione del primo di questi da parte dell'utente spinge l'`HomeAgent` alla creazione di un `PlayAgent`, che verrà descritto successivamente, e alla terminazione di sè stesso tramite il metodo `doDelete()`. Intuitivamente la pressione del tasto `btnLogout` causa l'evento che genera il ritorno all'interfaccia di login e alla creazione di un nuovo `LoginAgent`; questo evento riporta quindi il client allo stato di Login. La pressione di `btnAccountInfo`, infine, visualizza una nuova finestra contenente tutte le informazioni relative all'utente autenticato, tale finestra è gestita da un apposito agente `AccountInfoAgent`. L'unico behaviour (*CyclicBehaviour*) di questo agente attende, tramite semplice `receive()`, un `ACLMessage` conforme al *MessageTemplate* definito da `templateReturn`. Tale messaggio causa la riattivazione del `btnAccountInfo`, disattivato ad una precedente pressione. Questo accorgimento è necessario per garantire consistenza nella modifica dei dati e nelle operazioni di *Prelievo* e *Deposito* da parte dell'utente.

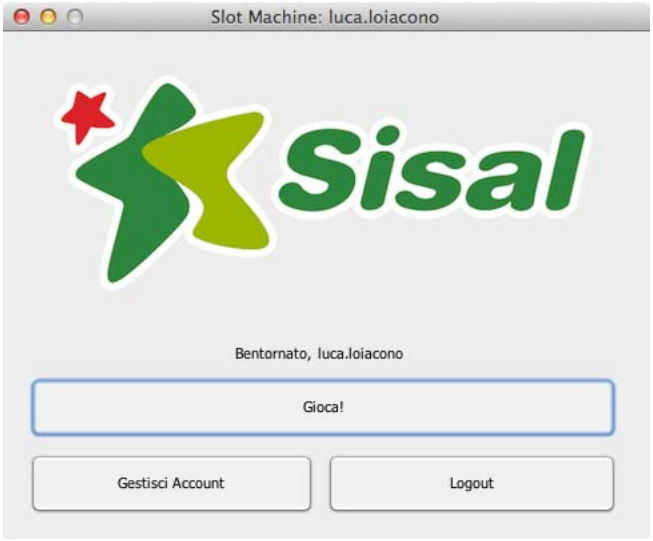

Figura 4.2: Client: Menù Iniziale.

4.2.3 AccountInfoAgent

Scopo di questo agente è permettere la visualizzazione delle informazioni relative all'account autenticato e la gestione dei movimenti di prelievo e deposito dal fondo associato all'utente. Questo agente comunica in maniera massiccia con il `DatabaseAgent`: inizialmente invia un primo *ACLMessage* con ontologia *infoutente* e contenente l'username dell'utente. Il server risponde a questa richiesta inviando due distinti *ACLMessage*:

1. `accountInfo_response`: Questo messaggio rispetta il template definito da `templateInfo`. Il contenuto di questo messaggio è un oggetto di tipo `entity.Utente` e specifica informazioni quali username, nome, cognome, saldo dell'utente (Figura 4.3);

2. `accountInfo_movimenti`: In questo caso il server risponde con un `ArrayList` di oggetti di tipo `entity.Movimento`. Il *MessageTemplate* è descritto da `templateMovimenti` e lo scopo di questo messaggio è creare una lista degli ultimi N movimenti (default 20) eseguiti dall'utente (Figura 4.3).

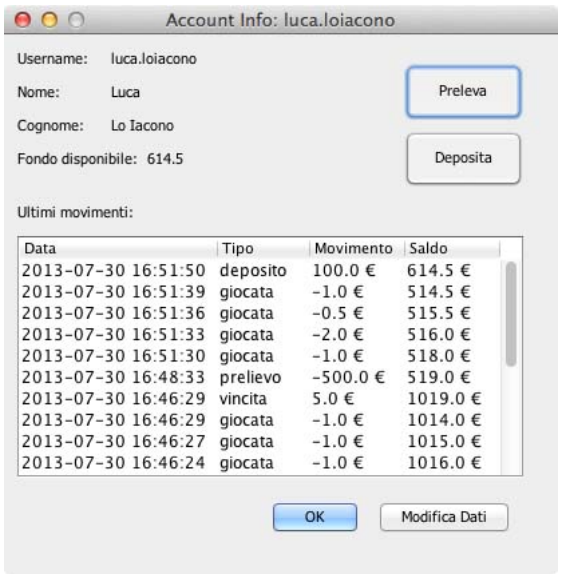

Figura 4.3: Client: Schermata Informazioni Account.

Gli eventi associati alla GUI di questo agente (`GUI.AccountInfo.java`) riguardano la pressione dei `JButton` `btnOk, btnPreleva, btnDeposita`. Il primo di questi predispone il ritorno all'`HomePanel` e al passaggio allo stato di `DELETED` dell'`AccountInfoAgent`. I bottoni `btnPreleva` e `btnDeposita` realizzano rispettivamente gli eventi associati alle funzioni di prelievo e deposito dal conto del titolare dell'account. In entrambi i casi, prima dell'avvio della comunicazione con il database, avviene un controllo del formato (`double`) dell'importo inserito dall'utente. Per quanto riguarda l'operazione di prelievo viene inoltre controllata la disponibilità del fondo a disposizione dell'utente. Una volta fatto ciò il client invia al server, quindi al database, un oggetto di tipo `movimento` contenente le specifiche del movimento che l'utente intende effettuare (data/ora, tipologia, importo, username, saldo aggiornato). Ad avvenuta risposta da parte del `DatabaseAgent`, la GUI comunica all'utente se l'operazione è andata a buon fine o meno (Figure 4.4, 4.5).

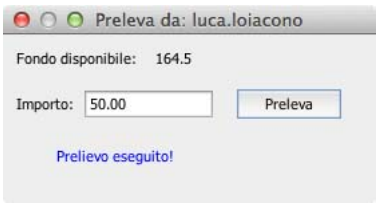

Figura 4.4: Client: Interfaccia prelievo avvenuto..

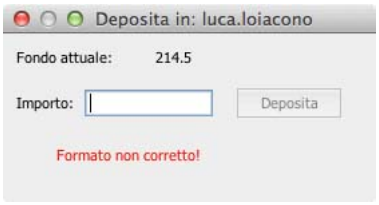

Figura 4.5: Client: Interfaccia deposito non avvenuto.

4.2.4 PlayAgent

Questo agente e la sua interfaccia grafica (`GUI.PlayGui.java`) sono il fulcro del sistema lato client. Tramite il `PlayAgent` l'utente può giocare ed, eventualmente, vincere. Ad apertura dell'interfaccia viene avviato il behaviour `ReceiveJackpotSaldo()` che specializza la classe `OneShotBehaviour`. Scopo di questo behaviour è interrogare il `DatabaseAgent` riguardo all'ammontare del saldo utente, del jackpot locale e del jackpot nazionale. Al fine di aggiornare costantemente gli importi sopra elencati il behaviour `ReceiveJackpotSaldo()` viene eseguito dall'agente dopo ogni singola giocata. In Figura 4.6 una rappresentazione dell'interfaccia di gioco:

Gli eventi contemplati da questo agente riguardano la pressione dei `JButton` `btnSpin` e `btnEsci`. Nel secondo caso c'è il rientro al menù principale, quindi la creazione di un nuovo `HomeAgent`, e la terminazione del `PlayAgent` corrente. Tramite i 3 `JRadioButton` (0.50 €, 1.00 €, 2.00 €) l'utente può impostare l'importo che intende puntare in una determinata giocata. Una volta attivato l'evento di *Spin* viene avviata la giocata: l'agente, tramite il behaviour `ComputaGiocata()`, prepara l'oggetto `giocata`, di tipo `entity.InfoGiocata`, impostando lo *username* del giocatore, l'*AID* del mittente e l'*importo* speci-

Figura 4.6: Client: Interfaccia di gioco.

ficato. Una volta fatto questo invia un messaggio ACL al server (`RicevitoreAgent`) e attende la risposta da parte di un `ElabGiocataAgent`. Questa risposta è accompagnata da una classica animazione composta dalla rotazione dei rulli della Slot Machine. In base alla composizione della risposta ricevuta, ed in particolare delle variabili `vincita`, `jackLoc` e `jackNaz`, viene comunicato a video l'esito della giocata e l'eventuale vincita di uno dei due jackpot previsti.

Figura 4.7: Client: Caso di vincita.

Figura 4.8: Client: Caso di non vincita.

Capitolo 5

Conclusioni e Sviluppi Futuri

Come è ben noto, il mercato mobile è in rapida espansione ed evoluzione, sempre più persone posseggono uno smartphone con piano dati attivato. Proprio per questo motivo si potrebbe in futuro cercare di creare un'applicazione mobile, il tutto per ampliare la clientela ed il volume delle giocate. Questa soluzione potrebbe essere resa ancora più semplice dalla presenza di una versione di Jade per Android, cioè *Jade Leap*. Attraverso questo framework si potrebbero riciclare varie parti del client PC ma si renderebbe in ogni caso necessaria una completa rivisitazione e realizzazione dell'interfaccia grafica per l'ambiente mobile. Il controllo dei dati relativi alle giocate potrebbe portare alla realizzazione di un sistema atto a realizzare studi e statistiche. Essi potrebbero essere molto utili per capire le abitudini dei giocatori (orari di gioco, importi medi delle giocate ecc.) e quindi migliorare l'esperienza di gioco o capire quando effettuare operazioni di marketing. Il sistema, come si può evincere analizzando lo schema ER, è già in parte predisposto a questo scopo. Analizzando la tabella Giocatori si può vedere la presenza del campo operatore, una figura che ha lo scopo di supervisionare il sistema e che ha accesso completo ai dati.