Hearts - Card game

Progetto di Architetture Software, A.A. 2010/2011

Realizzato da:

Vincenzo Alaia, vincenzomaria.alaia@studio.unibo.it

Carlo Donzelli, carlo.donzelli@studio.unibo.it

Davide Leonardi, davide.leonardi3@studio.unibo.it

Fabio Quinzi, fabio.quinzi@studio.unibo.it

Luca Valentini, luca.valentini7@studio.unibo.it

Introduzione

Scopo del documento

In questo documento verranno presentati le metodologie adottate e gli artefatti prodotti durante lo sviluppo del software Hearts. Questo software implementa il famoso e omonimo gioco di carte, in una versione giocabile online. Le regole adottate per realizzare la logica del gioco seguono quelle della variante più giocata in rete.

La parte iniziale del documento descrive i requisiti funzionali e non funzionali dell'applicazione. La seconda sezione descrive gli strumenti, le librerie e i framework utilizzati come ausilio per lo sviluppo del sistema. La terza parte del documento illustra, attraverso una serie di artefatti, l'architettura del sistema software. La sezione successiva descrive il processo di sviluppo utilizzato e spiega i motivi che hanno portato ad adottare tale metodologia. L'ultima parte del documento include un breve manuale d'uso dell'applicativo. In allegato viene consegnato un documento che raccoglie gli artefatti propri del processo di sviluppo utilizzato.

Specifica dei requisiti

Obiettivo del progetto

Obiettivo del progetto è la realizzazione di un sistema che permetta di partecipare a, e di seguire delle partite al gioco di carte Hearts. Secondo le specifiche rilasciate, l'interazione tra gli utenti può avvenire, a discrezione degli sviluppatori, attraverso un browser web o un client proprietario. Una volta autenticati, gli utenti hanno modo di vedere gli altri utenti collegati, il loro stato e il loro punteggio, i tavoli creati (avviati e non) e possono interagire fra loro con uno strumento di tipo chat. Gli utenti possono creare tavoli oppure aggregarsi a tavoli esistenti come giocatori o come spettatori.

Requisiti funzionali

- Il sistema deve fornire agli utenti la possibilità di registrarsi.
- Il sistema deve fornire un'interfaccia per l'utilizzo da parte di utenti previa autenticazione tramite meccanismo di login attraverso username e password.
- Il sistema deve fornire uno strumento di tipo chat che coinvolge tutti gli utenti connessi, sia nel tavolo che nella lobby.
- Il sistema deve fornire uno strumento che permetta agli utenti autenticati di visualizzare una lista di giocatori connessi e una lista di tavoli creati.
- Un utente libero deve avere la possibilità di creare un nuovo tavolo e di aggregarsi ad un tavolo già esistente (avviato o meno), sia come giocatore che come spettatore.
- Durante la fase di creazione del tavolo deve essere possibile selezionare il numero di utenti automatici (numero di bot compreso tra 0 e 3).
- Un utente in attesa ad un tavolo può in ogni momento abbandonare il tavolo e tornare libero. Se il giocatore che abbandona il tavolo è il creatore, il tavolo stesso viene distrutto e tutti i giocatori diventano liberi.
- Il sistema deve assegnare 4 punti al giocatore che ha vinto la manche e sottrarre 1 punto ai giocatori che hanno perso.

Requisiti non funzionali

- Il client deve essere eseguibile su almeno un Sistema Operativo. Opzionalmente può essere portabile su qualsiasi piattaforma.
- Il client deve essere facilmente fruibile da utenti di qualsiasi età e con conoscenze informatiche anche di basso livello.
- Il sistema deve sostenere l'interazione tra un numero di utenti nell'ordine delle decine.
- Il sistema deve essere disponibile 24 ore su 24.
- Il sistema deve funzionare correttamente, indipendentemente dalla presenza di sistemi Firewall o NAT sui nodi.
- L'utilizzo delle risorse dovrebbe essere ridotto.
- Il server dovrebbe essere interoperabile con altri client, sviluppati seguendo un protocollo definito.
- Le tecnologie utilizzate per la realizzazione del sistema devono essere di tipo opensource o quantomeno freeware.

Strumenti utilizzati

Per sviluppare l'architettura in oggetto sono stati utilizzati software per l'ausilio allo sviluppo, IDE, API e librerie, piattaforme. In questa sezione vengono descritti tali strumenti e vengono spiegate le motivazioni che hanno portato alla loro adozione.

IDE

Eclipse

Eclipse è un celebre *toolset* per lo sviluppo software. La scelta è ricaduta su questo IDE perché tutti i membri del team hanno una buona esperienza di utilizzo e perché lo ritengono uno strumento valido e completo.

Ausilio allo sviluppo

Omnigraffle

Omnigraffle è un software utile per disegnare tabelle e diagrammi. Disponibile esclusivamente per Mac OS X è stato scelto per la facilità di utilizzo, per la qualità degli artefatti prodotti, e perché 4 membri del team su 5 possiedono un Macintosh. Nello specifico il team ha utilizzato questo software per le sedute di brainstorming.

eUML2

eUML2 è un plugin per la modellazione UML in Eclipse che mette a disposizione tools per la creazione dei più comuni diagrammi (Class Diagram, Sequence Diagram, Use Case Diagram ecc.). La scelta del team è ricaduta su questo tool come naturale conseguenza all'adozione di Eclipse come ambiente di sviluppo. La perfetta integrazione tra i due software ha reso snella ed agile la modellazione del sistema.

Pages

Pages è un word processor prodotto da Apple. Destinato al sistema operativo Mac OS X, fa parte della suite di produttività iWork. Il team ha deciso di adottare questo software per la stesura della documentazione, visto che la maggior parte dei membri aveva già esperienza con questo prodotto, vista la facilità e l'immediatezza di utilizzo e vista l'elevata qualità degli artefatti prodotti con esso.

Piattaforme server

Openfire

Openfire è un server XMPP scritto in Java e rilasciato sotto licenza Apache. Facile da installare e da configurare, è caratterizzato da:

- portabilità
- pannello di amministrazione web-based
- interfaccia per plugin
- possibilità di personalizzazione
- connettività con database embedded o esterni.

Il team ha deciso di utilizzare questa piattaforma dopo aver definito i protocolli e lo stile architetturale da adottare per sviluppare il sistema.

API e Librerie

Smack

Smack è una libreria Java, opensource, client-side per sviluppare applicativi che fanno uso del protocollo XMPP. Include già delle estensioni per l'utilizzo del protocollo MUC (Multi-user Chat). Anche in questo caso il team ha deciso di utilizzare questa piattaforma dopo aver definito i protocolli e lo stile architetturale da adottare per sviluppare il sistema.

FrogX

Il progetto opensource FrogX è nato per fornire un componente esterno che abiliti l'utilizzo di servizi Multi-User Gaming su server XMPP. Queste librerie forniscono un supporto per la gestione di stanze, giocatori e spettatori. Il team ha deciso di utilizzare FrogX dopo aver definito i protocolli e lo stile architetturale da adottare per sviluppare il sistema e vista l'integrazione con la piattaforma Openfire.

JCardGame

JCardGame è una libreria per sviluppare giochi di carte. Include sia una parte di logica sia una parte di grafica. Il team ha deciso di utilizzare queste librerie solo per lo sviluppo della GUI del sistema, visto che forniscono i principali elementi che caratterizzano l'interfaccia utente di un semplice gioco di carte.

Architettura del software

In questa sezione verrà illustrata l'architettura del sistema software tramite degli artefatti utili a descrivere le scelte intraprese per lo sviluppo.

Innanzitutto la scelta è ricaduta sulla tipologia client-server, sia perché è notoriamente l'architettura più diffusa tra gli applicativi di questo genere e sia perché la presenza di un coordinatore centrale permette a numerosi nodi di condividere le risorse che esso offre, preoccupandosi della gestione degli accessi ad esse.

Nel sistema prodotto è stato utilizzato OpenFire, un noto server XMPP (Jabber) rilasciato sotto licenza opensource GPL. Il team ha deciso di adottare questo server per la solidità e la diffusione del protocollo XMPP, che si è dimostrato essere ideale per l'implementazione di uno strumento di chat, e particolarmente adatto per la gestione di stanze di gioco multiple ma indipendenti tra loro: il concetto di stanza di gioco è infatti facilmente assimilabile a quello di stanza di chat. La adozione del protocollo XMPP e del server OpenFire ha inoltre permesso di utilizzare una tecnica di server pushing, ideale sia per l'implementazione della chat sia per comunicare i nuovi stati di gioco ai vari client. Il server gestisce un database embedded chiamato HSLQDB.

Per facilitare la gestione delle stanze e dei relativi meccanismi, dei giocatori e degli spettatori, è stato utilizzato FrogX. Si tratta di un componente software aggiuntivo che offre inoltre un framework per la creazione di semplici giochi Multi-User. FrogX è perfettamente integrabile con OpenFire: dal pannello di controllo del server è possibile caricarlo come plugin.

Attraverso un' applicazione Java, realizzata utilizzando le API Smack, i nodi client possono collegarsi al server, utilizzare i servizi messi a disposizione dallo stesso e giocare al gioco di carte Hearts.

Casi d'uso

Di seguito sono elencati i casi d'uso del sistema:

- Registrazione di un utente.
- Autenticazione di un utente.
- Visualizzazione lista tavoli.
- Visualizzazione lista utenti.

- Creazione tavolo.
- Join tavolo come giocatore.
- Join tavolo come spettatore.
- Abbandona tavolo.
- Mossa di gioco.
- Chat tavolo.
- Chat lobby.

Registrazione di un utente

Descrizione
Per poter utilizzare il sistema, l'utente deve registrarsi.

Attori
Utenti non ancora registrati.

Precondizioni
Nessuna.

Azioni
L'utente inserisce lo username con cui desidera identificarsi nel sistema, la password scelta, il suo nome reale e il suo indirizzo email.

Autenticazione di un utente

Descrizione
Per poter utilizzare il sistema, un utente registrato deve autenticarsi.

Attori
Utenti registrati, ma non autenticati.

Precondizioni
L'utente deve essere già registrato al sistema

Azioni
L'utente inserisce il suo username e la sua password per autenticarsi al sistema

Visualizzazione lista tavoli

Descrizione
L'utente desidera visualizzare la lista dei tavoli attivi e non.

Attori
Utenti autenticati.

Precondizioni
L'utente deve essere autenticato al sistema.

Azioni
L'utente accede alla view che mostra la lista dei tavoli attivi e non.

Visualizzazione lista utenti

Descrizione
L'utente desidera visualizzare la lista degli utenti connessi al sistema.

Attori
Utenti autenticati.

Precondizioni
L'utente deve essere autenticato al sistema.

Azioni
L'utente accede alla view che mostra la lista degli utenti connessi al sistema.

Creazione tavolo

Descrizione
L'utente desidera creare un nuovo tavolo da gioco

Attori
Utenti autenticati liberi.

Precondizioni
L'utente deve essere autenticato al sistema.

Azioni
L'utente accede alla view per creare il nuovo tavolo da gioco. Inserisce il numero di bot che desidera inserire nel gioco.

Join tavolo come giocatore

Descrizione
L'utente si collega ad un tavolo esistente e non ancora avviato.

Attori
Utenti autenticati liberi.

Precondizioni
L'utente deve essere autenticato al sistema.

Azioni
L'utente seleziona il tavolo non ancora avviato a cui collegarsi. Se, dopo il suo ingresso, il tavolo è completo allora il match ha inizio. Altrimenti l'utente rimarrà in stato di attesa prima che si colleghino i giocatori rimanenti.

Mossa di gioco

Descrizione
L'utente effettua una mossa secondo le regole del gioco

Attori
Utenti autenticati, che sono collegati come giocatori ad un tavolo del sistema.

Precondizioni
L'utente deve essere autenticato, deve essere collegato come giocatore al tavolo, deve essere il suo turno di gioco.

Azioni
L'utente effettua una mossa secondo le regole del gioco. Nel caso in cui si tratti del primo turno allora selezionerà le 3 carte da passare al suo avversario di sinistra. Nel caso in cui si tratti di un turno di gioco standard allora l'utente sceglierà la carta da giocare.

Chat tavolo

Descrizione
L'utente comunica con altri utenti connessi al suo stesso tavolo.

Attori
Utenti autenticati e connessi ad un medesimo tavolo.

Precondizioni
L'utente deve essere autenticato e connesso ad un tavolo

Azioni

L'utente invia messaggi e ne riceve dagli utenti connessi al suo medesimo tavolo.

Chat lobby

Descrizione

L'utente comunica con altri utenti connessi al sistema.

Attori

Utenti autenticati e liberi.

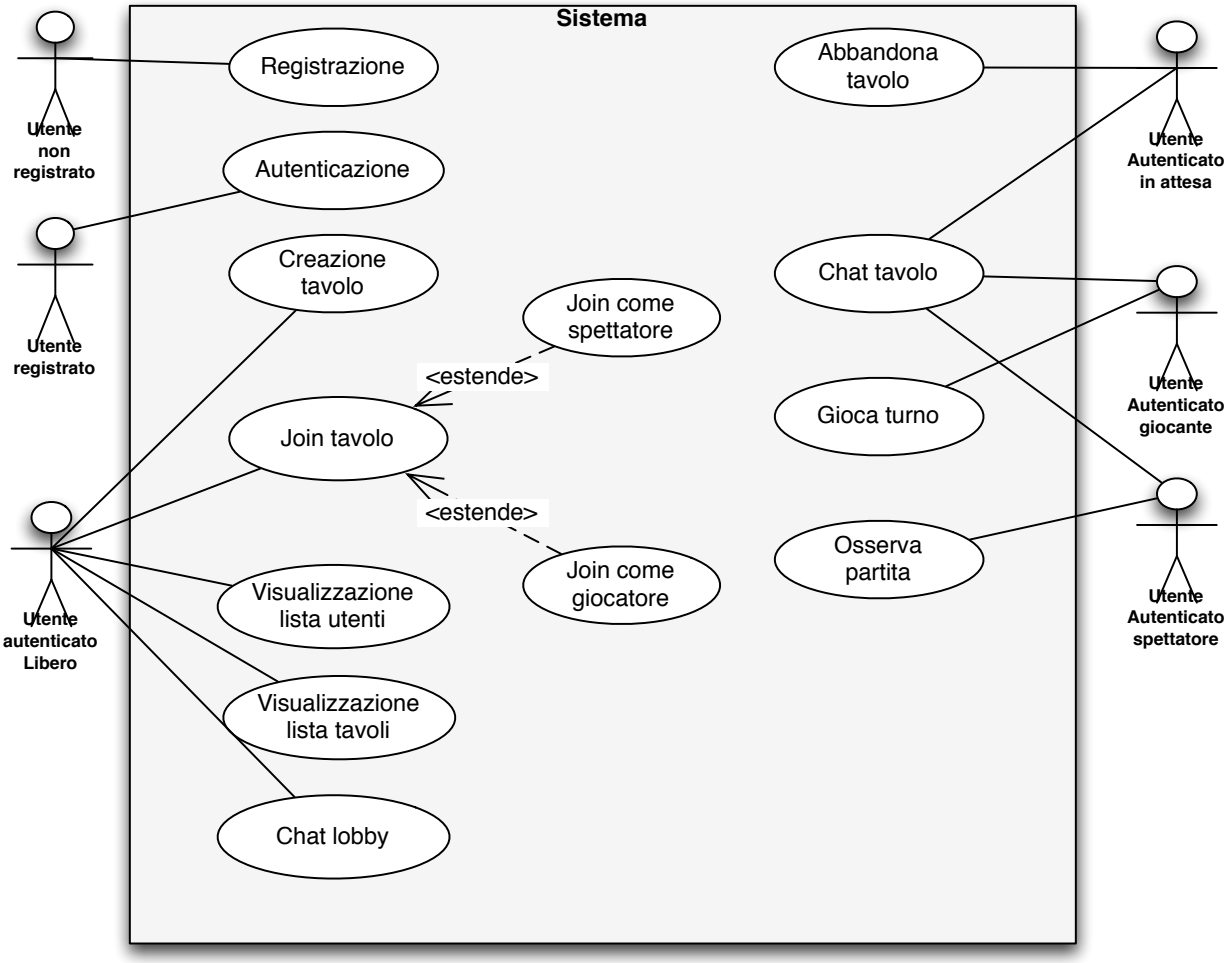
Precondizioni

L'utente deve essere autenticato.

Azioni

L'utente invia messaggi e ne riceve dagli utenti connessi al sistema e liberi.

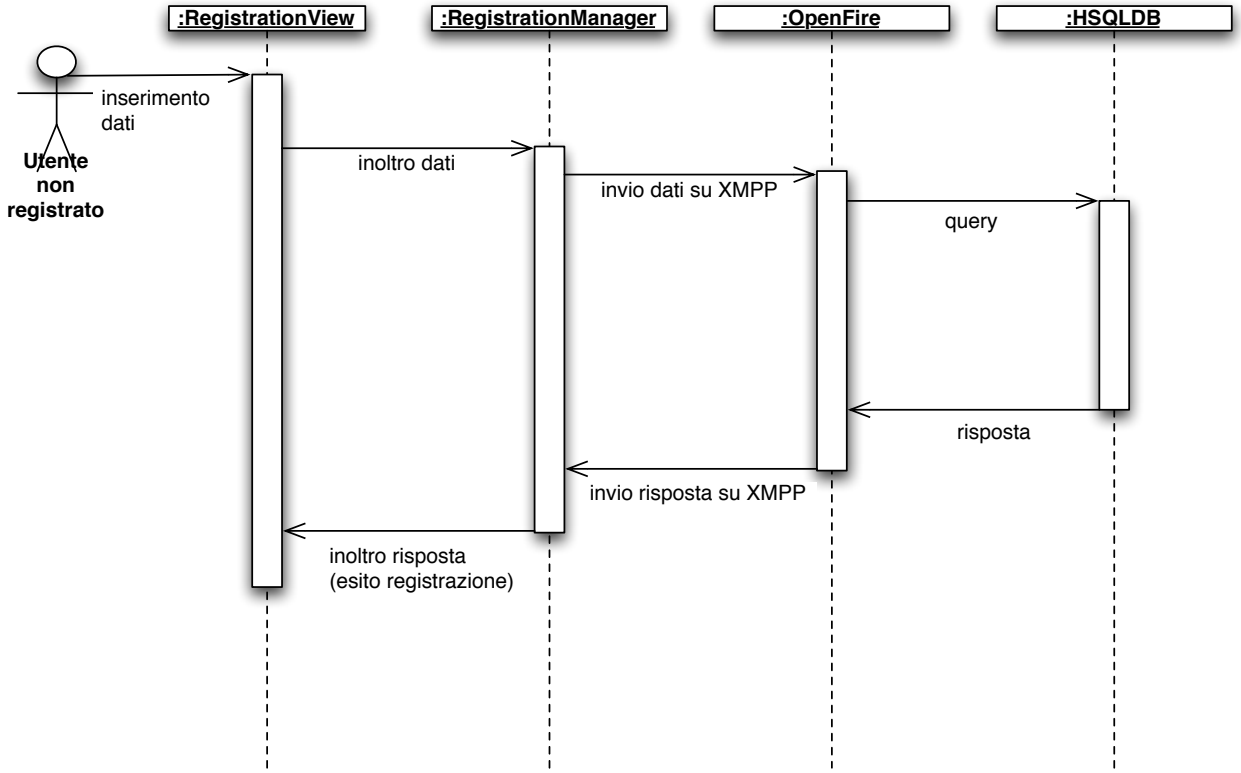
Diagramma completo dei casi d'uso



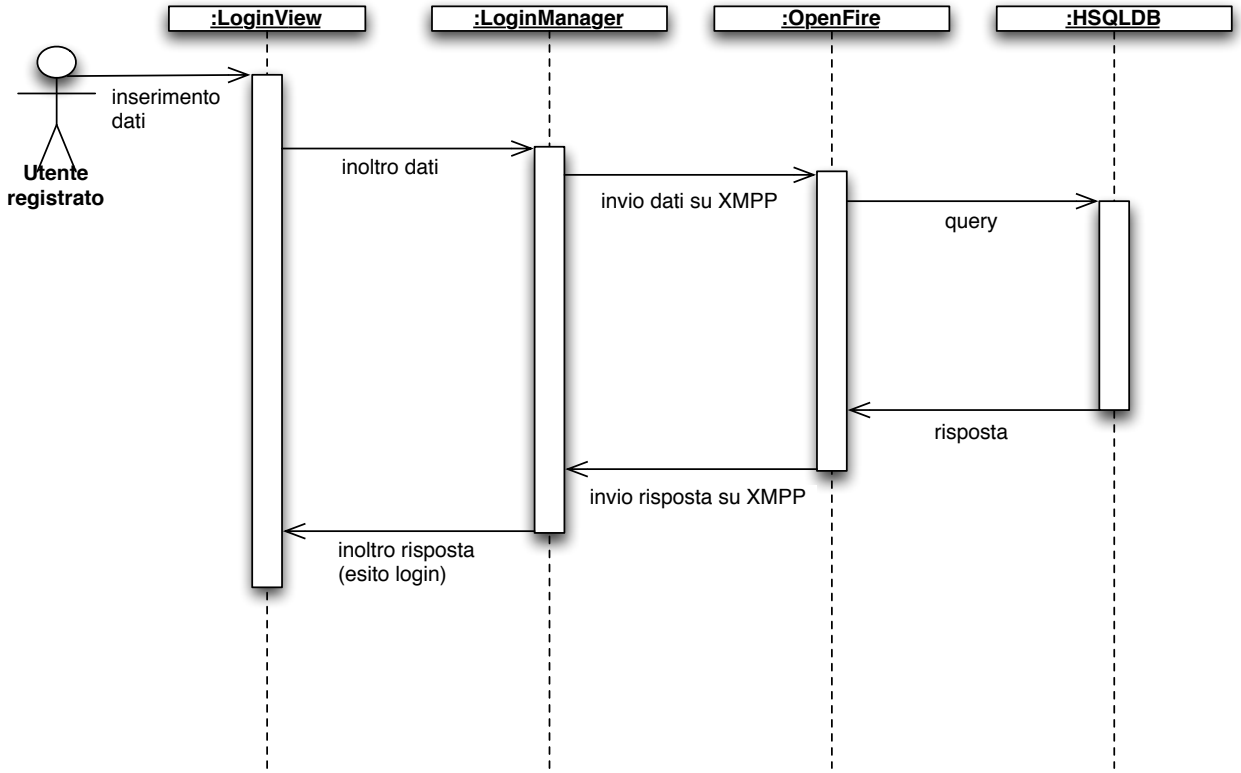
Diagrammi di sequenza

Questa sezione presenta alcuni diagrammi di sequenza riguardanti alcune delle interazioni tra i componenti software stessi e l'utente.

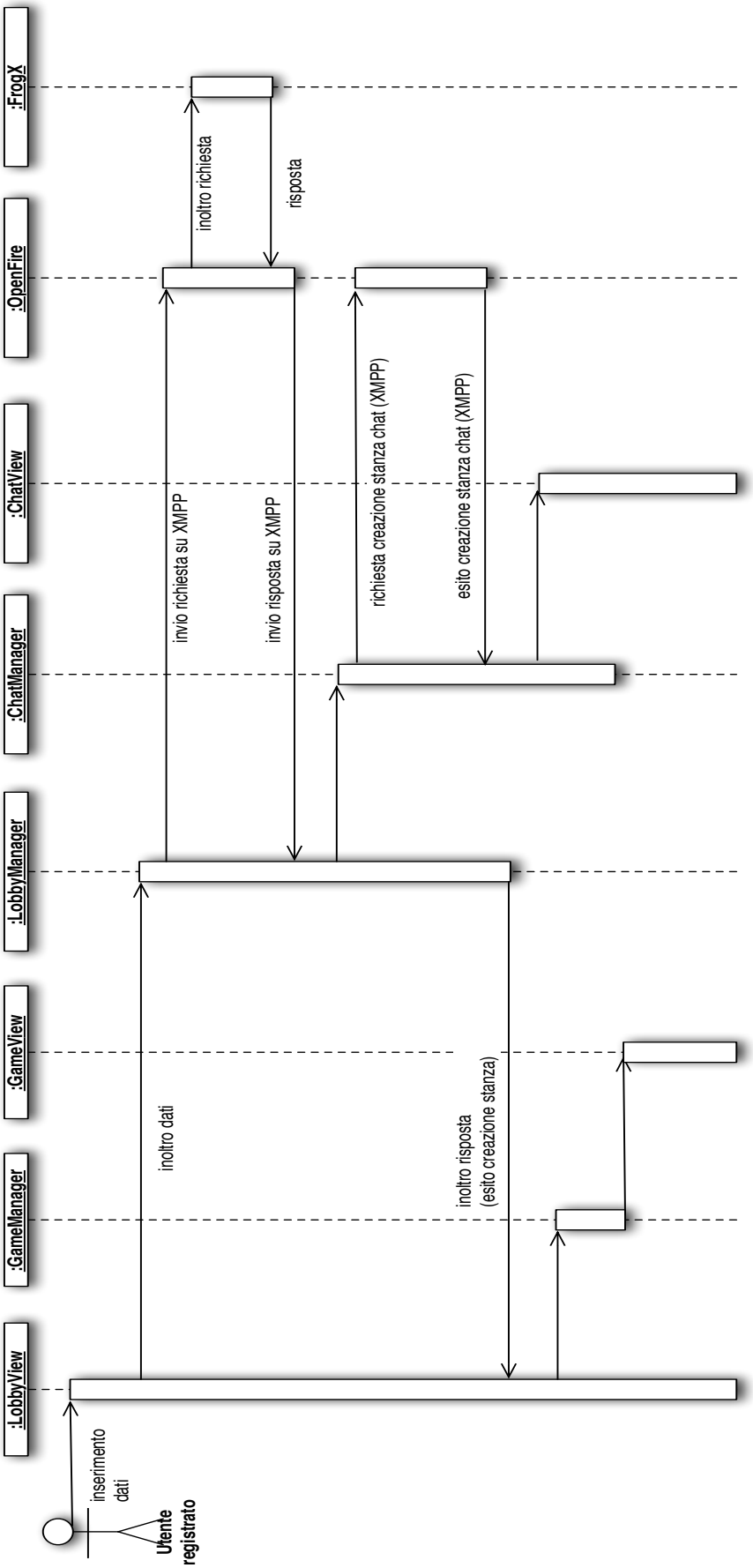
Registrazione



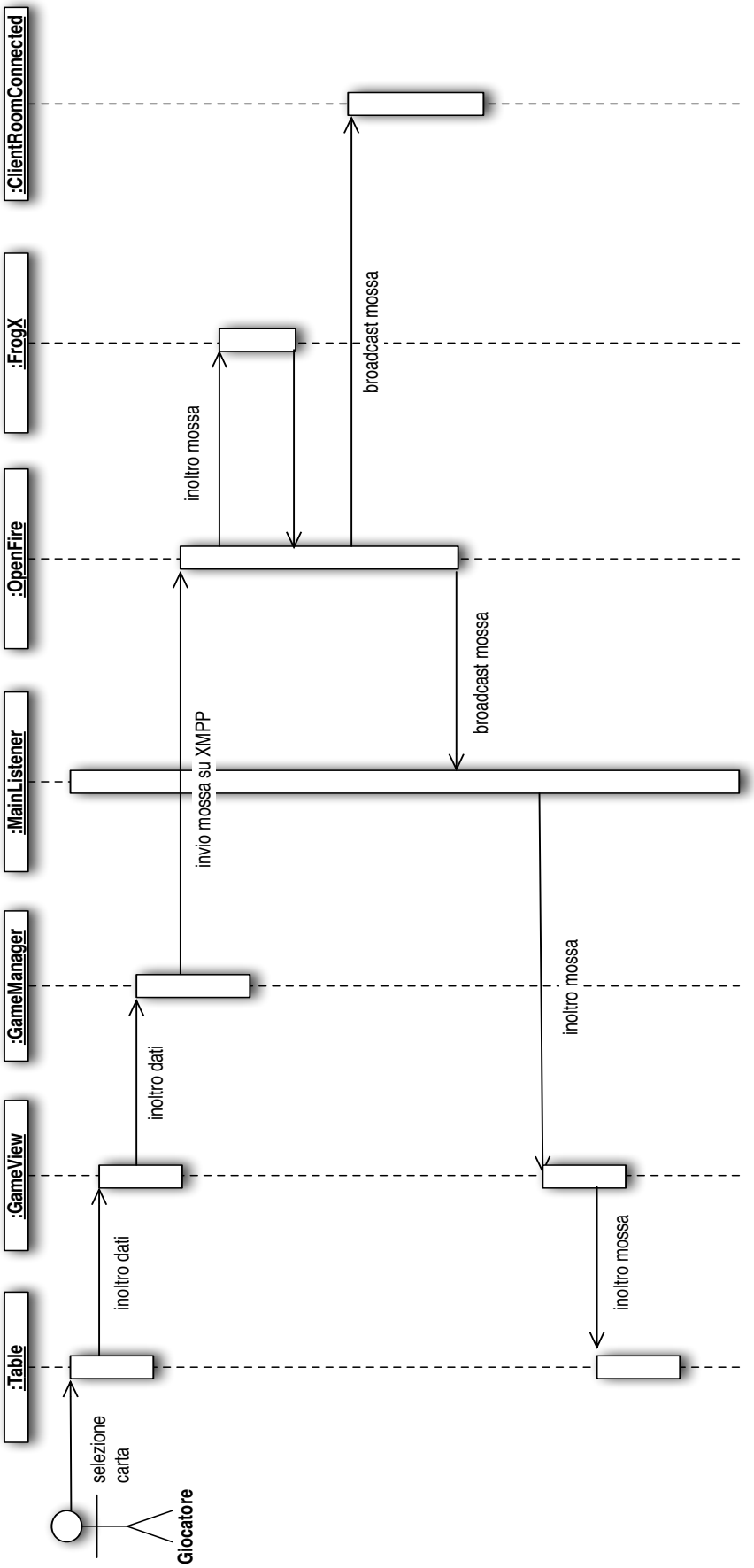
Login

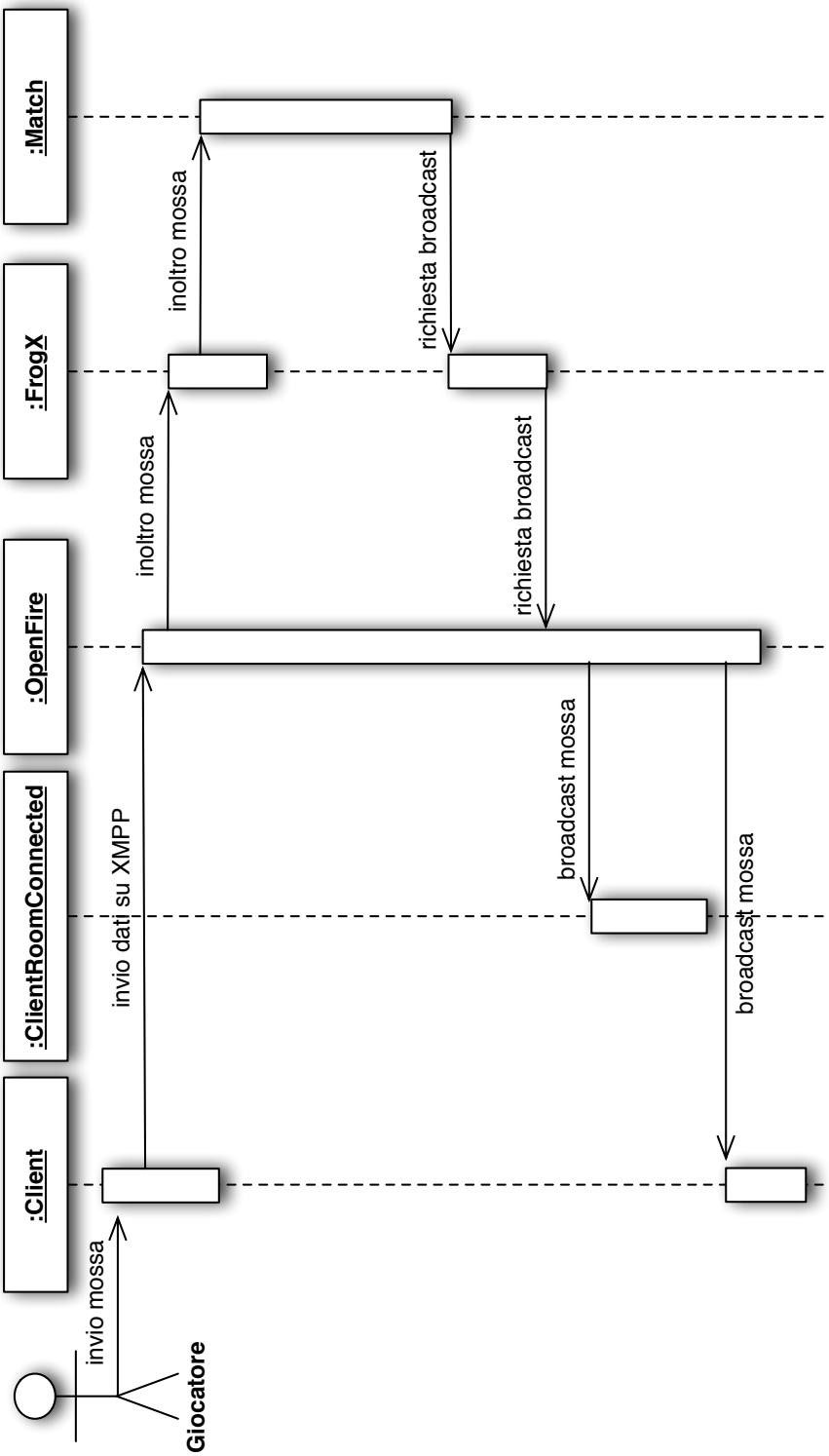

:LoginView
:LoginManager
:OpenFire
:HSQLDB
Utente registrato
inserimento dati
inoltro dati
invio dati su XMPP
query
risposta
invio risposta su XMPP
inoltro risposta (esito login)

Creazione stanza

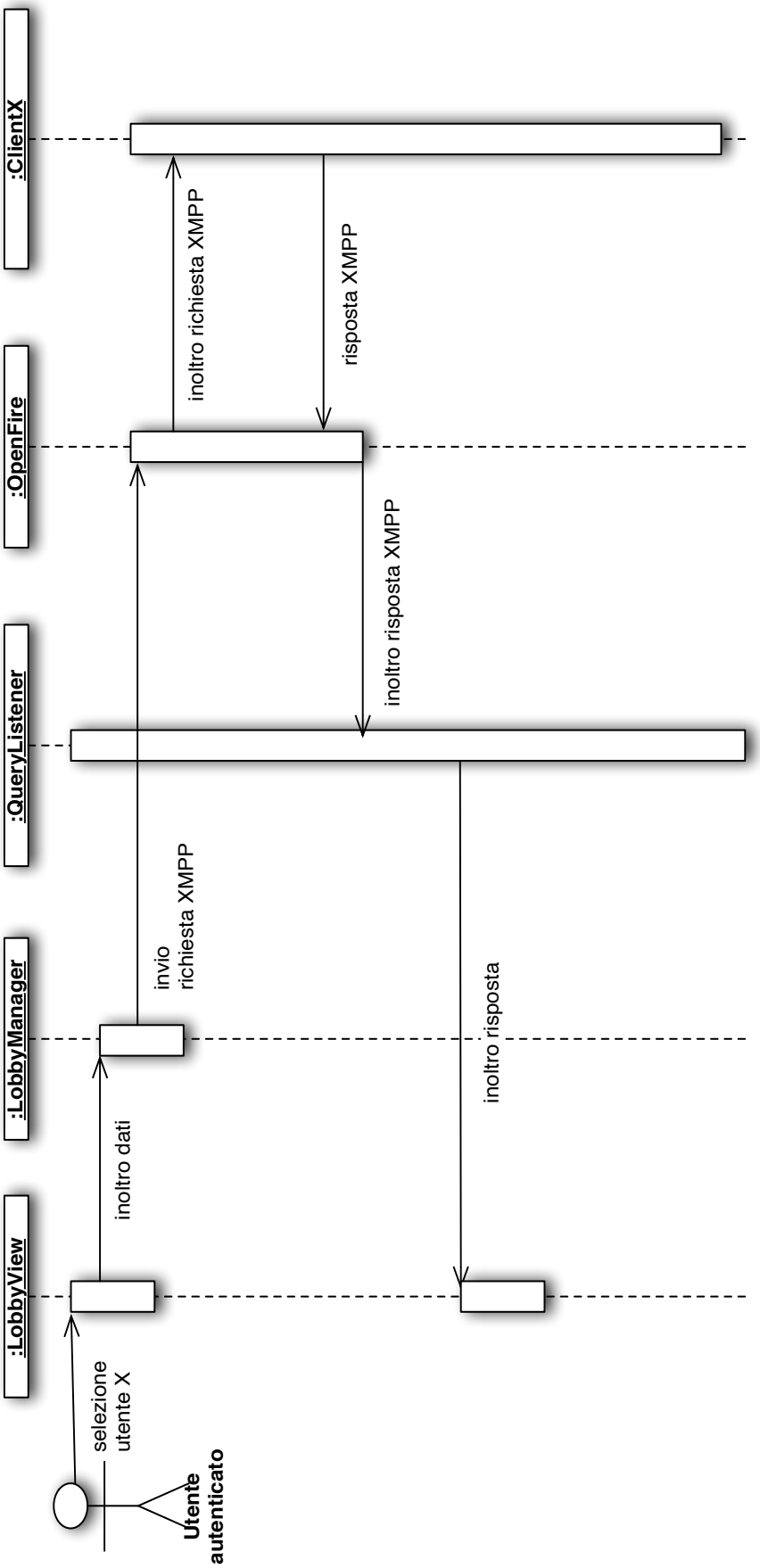


Gioca carta

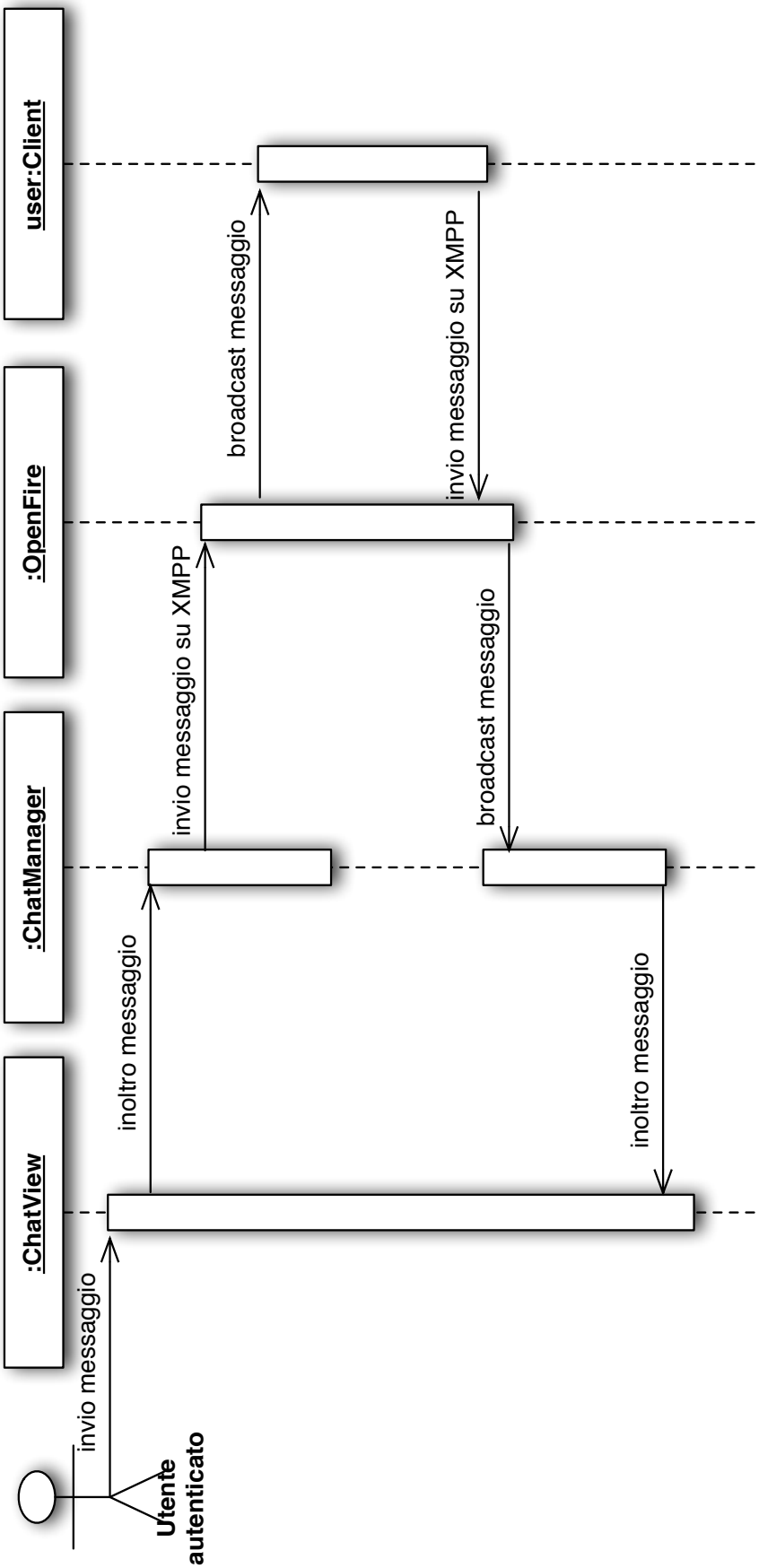

:ClientRoomConnected
:FrogX
:OpenFire
:MainListener
:GameManager
:GameView
:Table
Giocatore
selezione carta
inoltro dati
inoltro dati
invio mossa su XMPP
inoltro mossa
broadcast mossa
inoltro mossa
broadcast mossa
inoltro mossa


:Match
:FrogX
:OpenFire
:ClientRoomConnected
:Client
Giocatore
invio mossa
invio dati su XMPP
inoltro mossa
inoltro mossa
richiesta broadcast
richiesta broadcast
broadcast mossa
broadcast mossa

Richiesta informazioni utente



Invio e ricezione dei messaggi di chat



Diagrammi dei package

In questa sezione vengono mostrati i diagrammi dei package relativi al sistema sviluppato.

La struttura del sistema lato Client prevede l'utilizzo di un pattern MVC, il quale è basato sulla separazione dei compiti fra i componenti software che interpretano tre ruoli principali:

- il *model* fornisce i metodi per accedere ai dati utili all'applicazione;
- il *view* visualizza i dati contenuti nel model e si occupa dell'interazione con utenti e agenti;
- il *controller* riceve i comandi dell'utente (in genere attraverso il view) e li attua modificando lo stato degli altri due componenti

Questo schema, fra l'altro, implica anche la tradizionale separazione fra la logica applicativa (in questo contesto spesso chiamata "logica di business"), a carico del *controller* e del *model*, e l'interfaccia utente a carico del *view*.

Nel package listeners sono contenute le classi che si occupano dell'ascolto dei messaggi XMPP provenienti dal server. Questo consente di adottare un modello di programmazione *event-oriented* .

Le classi contenute nel package *providers* forniscono metodi e strutture dati per il parsing delle estensioni del protocollo XMPP.

Infine nel package *utils* sono contenute classi di importanza minore rispetto a quelle precedentemente elencate ma comunque necessarie per lo svolgimento di operazioni di utilità a livello di programmazione.

Per quanto riguarda il lato server, per una questione di praticità, si è deciso di raccogliere tutte le classi sviluppate in un solo package. Per una migliore comprensione dell'architettura del server rimandiamo alla lettura del diagramma delle classi presente nella relativa sezione.

Diagramma package client

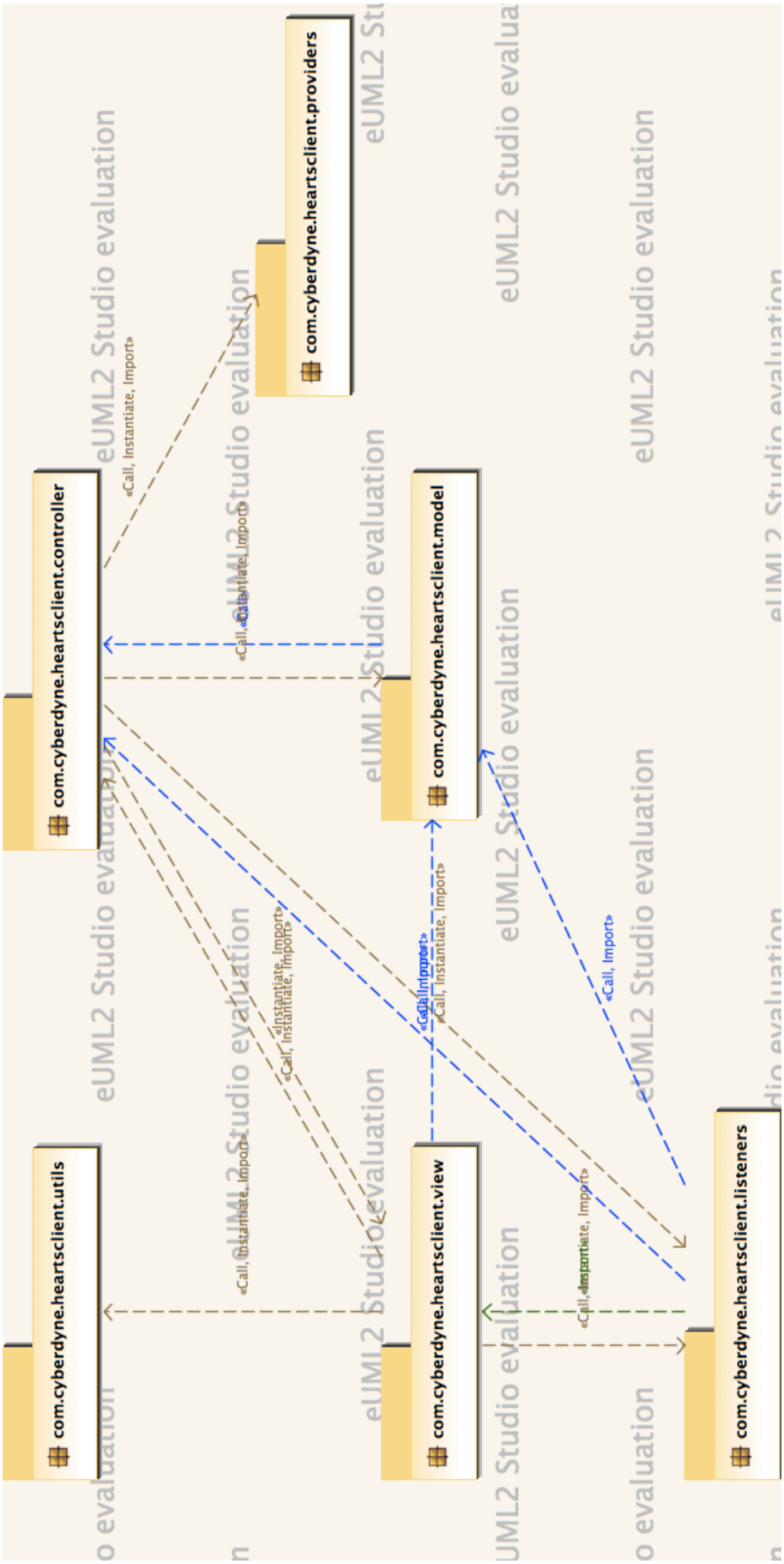
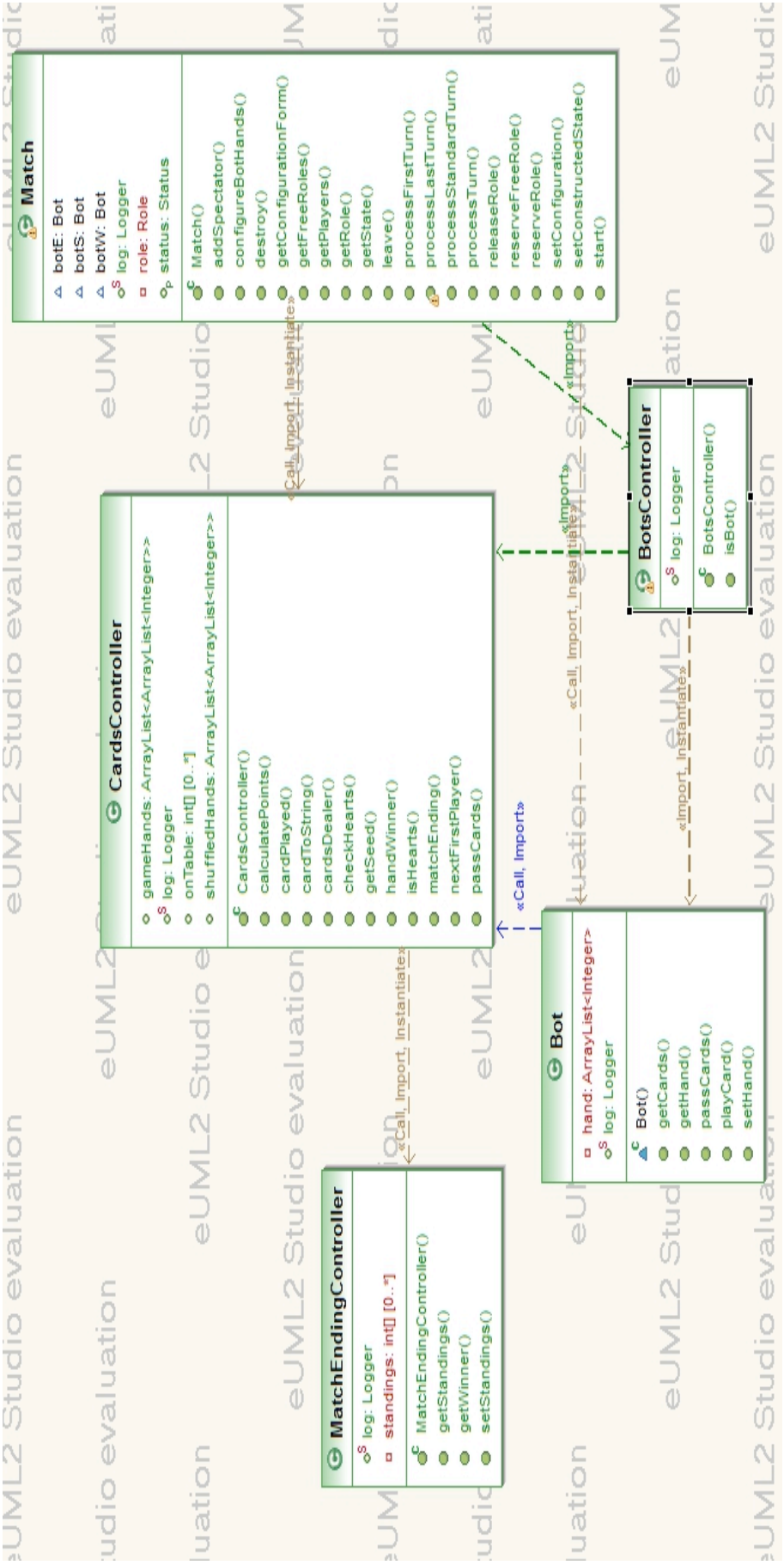


Diagramma delle classi server



CardsController
Questa classe si occupa della gestione delle regole del gioco Hearts, come ad esempio la regola del 2 di fiori e dei cuori infranti. Inoltre esegue la distribuzione delle carte di gioco, controlla le carte che sono giocabili e calcola il giocatore successivo.

MatchEndingController
Questa classe ha come unica funzione la gestione dell'ultimo turno di gioco e, una volta terminata la partita corrente calcola il vincitore e i punteggi di tutti i giocatori.

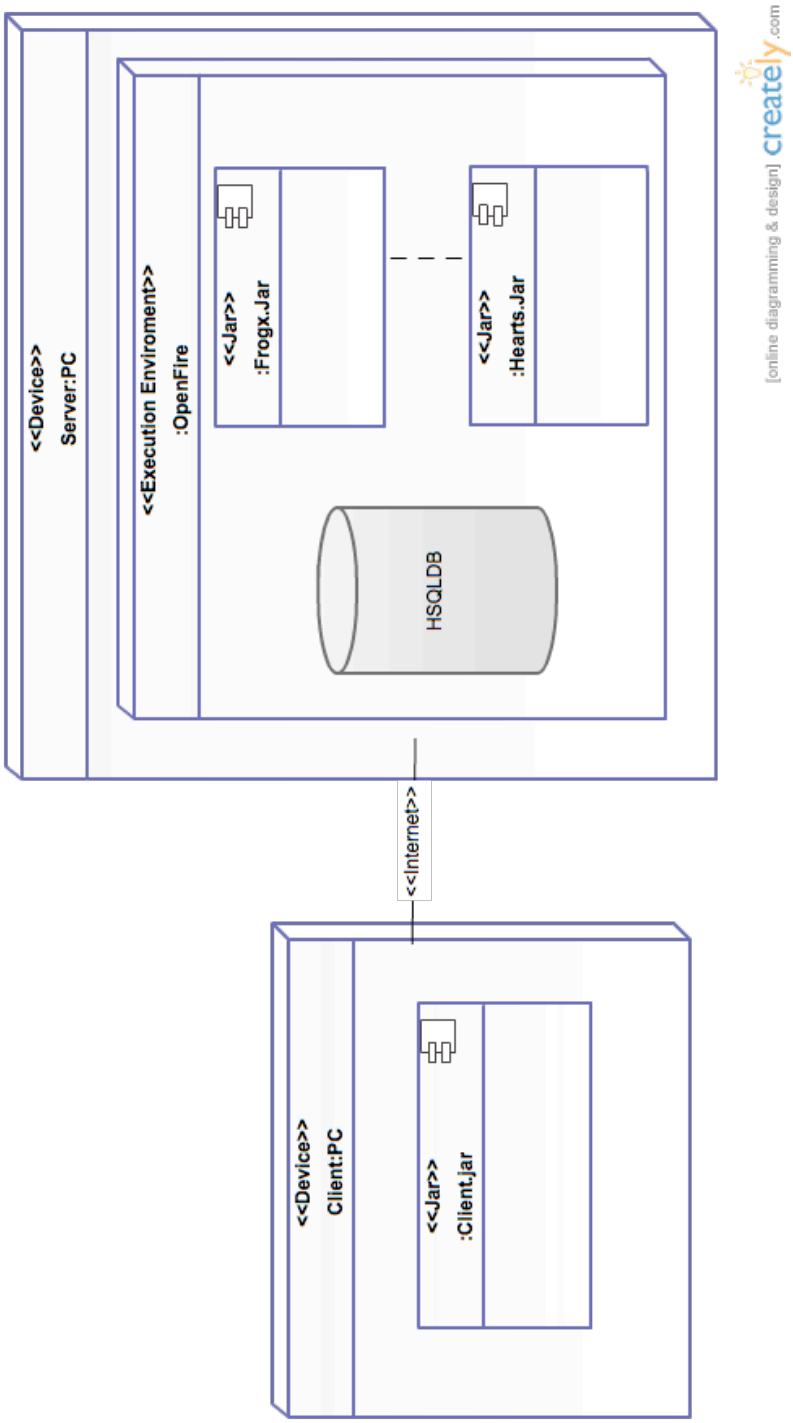
Bot
Questa classe contiene le strutture dati dei bot, che comprendono le loro mani di gioco e le primitive per controllare le loro carte (sia lanciate e sia passate ad altri giocatori) e per fargliele giocare.

BotsController
Questa classe è l'interfaccia delle classe Bot, necessaria per la comunicazione della classe Match con la classe Bot.

Match
Questa classe si occupa della gestione dei turni, riceve i messaggi contenenti le carte giocate dai client, interpreta il messaggio di creazione della stanza, assegna i ruoli di gioco ai partecipanti alla partita, gestisce la logica di gioco come i ruoli, i nomi, le carte in mano e il tavolo di gioco, gli spettatori.

Diagramma di deployment

Di seguito è riportato il diagramma di deployment del sistema software sviluppato.



Nello specifico il client proprietario, un applicativo java, si interfaccia con una singola unità (fisica) server dove vi è in esecuzione un'istanza del server XMPP Openfire utilizzato. FrogX.jar e Hearts.jar sono due plugin di Openfire: il primo è un componente esterno che abilita l'utilizzo di servizi MUG (Multi-User Gaming) su server XMPP; il secondo è un estensione di FrogX stesso (cioè si avvale dei servizi di gestione stanze, turni, giocatori e spettatori che esso fornisce) che implementa buona parte della logica dell'applicazione.

Protocollo adottato

Il protocollo adottato per la realizzazione del sistema è un'estensione di XMPP-MUG http://www.frogx.org/xep/multi-user_gaming.html

Autenticazione

Client invia richiesta di autenticazione

<auth mechanism="DIGEST-MD5" xmlns="urn:ietf:params:xml:ns:xmpp-sasl"></auth>

Server risponde con un challenge

<challenge xmlns="urn:ietf:params:xml:ns:xmpp-sasl">

cmVhbG09InRhdXJvbiIsbm9uY2U9IjdROHhLQWpCL3hZZFE2ZmJITFQ1c3FuZDZ1Um94Vml4a21aRUR0RWwiLHFvcD0iYXV0aCIsY2hhcnNldD11dGYtOCxhbGdvcml0aG09bWQ1LXNlc3M=

</challenge>

Client risponde con la "soluzione" del challenge

<response xmlns="urn:ietf:params:xml:ns:xmpp-sasl">

Y2hhcnNldD11dGYtOCx1c2VybmFtZT0idmluc3AiLHJlYWxtPSJ0YXVyb24iLG5vbmNlPSI3UTh4S0FqQi94WWRRNmZiSExUNXNxbmQ2dVJveFZiOGttWkVEdEVslixuYz0wMDAwMDAwMSxjbm9uY2U9ImdWakJoV01HZEtpSy80WXF1Q0REejhhMjN0NVBqZzRCRHhabml4dnliLGRpZ2VzdC11cmk9InhtcHAvdGF1cm9uIixtYXhidWY9NjU1MzYscmVzcG9uc2U9YTlkMTY0N2NmNjdkZmNlYTg1Mjk5MThjN2UwN2I2NjcscW9wPWF1dGgsYXV0aHppZD0idmluc3Ai

</response>

Server risponde con l'esito della richiesta di autenticazione

<success xmlns="urn:ietf:params:xml:ns:xmpp-sasl">

cnNwYXV0aD1hNzg3ZjhiYjMxNjYxYjhhY2ExZGNmODdlODViZDhiMQ==

</success>

Creazione stanza di gioco e avvio partita

Il client invia il messaggio di richiesta di creazione stanza di gioco
(lo stesso messaggio viene utilizzato per il join nella stanza)

```
<presence xmlns="http://jabber.org/protocol/disco#items" id="NY1J0-20"
    to="stanza@hearts.tauron/     vinsp" from="vinsp@tauron">

    <game xmlns="http://jabber.org/protocol/mug" var="http://jabber.org/protocol/mug/hearts"/>

</presence>
```

Server risponde con stato della stanza

```
<presence from="stanza@hearts.tauron" to="vinsp@tauron/Smack">

    <game xmlns="http://jabber.org/protocol/mug">

        <status>created</status>

        <state xmlns="http://jabber.org/protocol/mug/hearts">

            <bots>0</bots>

            <turn>north</turn>

            <table/>

        </state>

    </game>

</presence>
```

E il ruolo del creatore che è, ovviamente, owner

```
<presence xmlns="http://jabber.org/protocol/disco#items" id="NY1J0-20" to="vinsp@tauron/Smack"
    from="stanza@hearts.tauron/vinsp">

    <game xmlns="http://jabber.org/protocol/mug">

        <item affiliation="owner" role="north"/>

    </game>

</presence>
```

L'owner invia la configurazione della stanza

```
<iq id="create2" to="stanzapanza@hearts.tauron" from="vinsp@tauron" type="set">
    <query xmlns="http://jabber.org/protocol/mug#owner">
        <options>
            <x xmlns="jabber:x:data" type="submit">
                <field var="FORM_TYPE">
                    <value>http://jabber.org/protocol/mug#matchconfig</value>
                </field>
                <field var="mug#roomconfig_roomname">
                    <value>stanzapanza</value>
                </field>
                <field var="mug#roomconfig_matchdesc">
                    <value></value>
                </field>
                <field var="mug#roomconfig_moderated">
                    <value>0</value>
                </field>
                <field var="mug#roomconfig_allowinvites">
                    <value>0</value>
                </field>
                <field var="mug#roomconfig_maxusers">
                    <value>10</value>
                </field>
                <field var="mug#roomconfig_publicroom">
                    <value>1</value>
                </field>
                <field var="mug#roomconfig_membersonly">
```

```
                    <value>0</value>
                </field>
                <field var="mug#roomconfig_anonymity">
                    <value>non-anonymous</value>
                </field>
                <field var="mug#roomconfig_passwordprotectedroom">
                    <value>0</value>
                </field>
            </x>
        <options xmlns="http://jabber.org/protocol/mug/hearts">
            <x xmlns="jabber:x:data" type="submit">
                <field var="FORM_TYPE">
                    <value>http://jabber.org/protocol/mug/hearts#heartsconfig
                    </value>
                </field>
                <field var="mug/hearts#config_bot">
                    <value>2</value>
                </field>
            </x>
        </options>
        </options>
    </query>
</iq>
```

Il server risponde con il nuovo stato

<presence from="stanza@hearts.tauron" to="vinsp@tauron/Smack">

<game xmlns="http://jabber.org/protocol/mug">

<status>inactive</status>

<state xmlns="http://jabber.org/protocol/mug/hearts">

<bots>2</bots>

<turn>north</turn>

<table/>

</state>

</game>

</presence>

Presence nuovo giocatore

Ogni qual volta si connette un giocatore alla stanza, i giocatori connessi ricevono un messaggio di "presence" che li notifica dell'avvenuta connessione.

<presence xmlns="http://jabber.org/protocol/disco#items" id="6vH8d-15" to="vinsp@tauron/Smack" from="stanzaprova@hearts.tauron/davidep">

<game xmlns="http://jabber.org/protocol/mug">

<item affiliation="none" role="east" jid="davidep@tauron/Smack"/>

</game>

</presence>

Inizio partita

Quando saranno connessi quattro giocatori, lo stato della stanza passerà ad "attive" e saranno assegnate le carte a ciascun giocatore

```
<presence from="stanza@hearts.tauron" to="vinsp@tauron/Smack">
    <game xmlns="http://jabber.org/protocol/mug">
        <status>active</status>
        <state xmlns="http://jabber.org/protocol/mug/hearts">
            <bots>3</bots>
            <turn>north</turn>
            <table>
                <playerN Card1="8" Card2="20" Card3="21" Card4="30" Card5="31"
                Card6="34" Card7="36" Card8="37" Card9="38" Card10="42" Card11="43"
                Card12="50" Card13="51" ontable="0"/>
                <playerE Card1="1" Card2="2" Card3="3" Card4="5" Card5="7" Card6="9"
                Card7="15" Card8="17" Card9="24" Card10="32" Card11="45"
                Card12="46" Card13="48" ontable="0"/>
                <playerS Card1="6" Card2="12" Card3="16" Card4="23" Card5="26"
                Card6="29" Card7="33" Card8="35" Card9="39" Card10="41" Card11="44"
                Card12="47" Card13="49" ontable="0"/>
                <playerW Card1="4" Card2="10" Card3="11" Card4="13" Card5="14"
                Card6="18" Card7="19" Card8="22" Card9="25" Card10="27" Card11="28"
                Card12="40" Card13="52" ontable="0"/>
            </table>
        </state>
    </game>
</presence>
```

Invio carta giocata

<message xmlns="http://jabber.org/protocol/disco#items" id="Bsps2-19"
to="stanzaprova@hearts.tauron" from="vinsp@tauron" type="chat">

<turn xmlns="http://jabber.org/protocol/mug#user">

<move xmlns="http://jabber.org/protocol/mug/hearts" turntype="standard" card1="13"
card2="" card3=""/>

</turn>

</message>

Il messaggio viene inoltrato dal server a tutti gli occupanti della stanza di gioco

Chat

Creazione stanza di chat

Avviene in maniera del tutto simile alla creazione della stanza di gioco, ma viene utilizzato il servizio "conference" anziché "herts"

Invio messaggio di chat

<message xmlns="http://jabber.org/protocol/disco#items" id="Bsps2-12"
to="lobby@conference.tauron" type="groupchat">

<body>messaggio di chat di prova</body>

</message>

Il messaggio viene inoltrato dal server a tutti gli occupanti della stanza di chat

Richiesta informazioni utente

<iq id="disco2" to="lobby@conference.tauron/vinsp" from="lobby@conference.tauron/davidep"
type="get">

<query xmlns="http://jabber.org/protocol/disco#items"></query>

</iq>

Il server inoltra la richiesta all'utente in questione che risponde al server stesso.Il server si occupa di inoltrare la risposta al richiedente.

<iq id="disco2" to="davidep@tauron/Smack" from="lobby@conference.tauron/vinsp" type="result">

<query xmlns="http://jabber.org/protocol/disco#items" status="libero" score="3"/>

</iq>

Processo di sviluppo

Metodologia di sviluppo

Per sviluppare l'architettura in oggetto, i membri del team hanno deciso di adottare la metodologia *Scrum*. Nelle seguenti sezioni vengono descritte le motivazioni che hanno portato verso questa scelta, vengono illustrati il funzionamento di tale metodologia e gli artefatti prodotti e vengono elencati i ruoli definiti all'interno del team.

Cos'è Scrum

Scrum è una metodologia agile di sviluppo del software che prevede di dividere il progetto in blocchi rapidi di lavoro (sprint) alla fine dei quali è possibile consegnare una versione funzionante del software al cliente. Scrum indica come definire i dettagli del lavoro da svolgere nell'immediato futuro (backlog) e organizza riunioni giornaliere del team di sviluppo (daily scrum) per verificare l'andamento dello sviluppo del software.

Perchè Scrum

Il team di sviluppo ha deciso di utilizzare la metodologia Scrum:

1. per la facilità di adozione;
2. per l'efficacia nel definire i dettagli del lavoro da svolgere nell'immediato futuro;
3. per il metodo di gestione dell'interazione tra i componenti del team (il team è autogestito);
4. per la sua leggerezza;
5. per la possibilità di adeguarsi facilmente ai cambiamenti delle specifiche del sistema.

Ruoli

Ruoli Core

Nome ruolo	Funzioni	Incaricato/i
Scrum Master	gestisce documentazione, meeting, garantisce il funzionamento della metodologia SCRUM	Fabio Quinzi
Product Owner	colui che scrive e gestisce il Product Backlog	Vincenzo Alaia
Team	team di sviluppo autogestito	Fabio Quinzi, Vincenzo Alaia, Carlo Donzelli, Luca Valentini, Davide Leonardi

Ruoli Ausiliari

Nome ruolo	Funzioni	Incaricato/i
Manager	colui che si occupa della gestione dell'infrastruttura, delle tecnologie e dell'ambiente di sviluppo per l'intero progetto	Luca Valentini
Stakeholders	sono coloro che durante le fasi di sprint review si occupano di controllare che le funzionalità del prodotto siano allineate a quelle previste	Fabio Quinzi, Vincenzo Alaia, Carlo Donzelli, Luca Valentini, Davide Leonardi

Meetings

Daily Scrum:

1. Organizzazione meeting
    1.1. Il meeting inizia esattamente ad una certa ora prefissata.
    1.2. Il meeting dura massimo 15 minuti.
    1.3. Il meeting ricorre sempre alla stessa ora e nello stesso posto tutti i giorni di lavoro. Nel caso non sia possibile riunire il team, il meeting avviene in videoconferenza
2. Durante il meeting ogni membro risponde alle seguenti domande
    2.1. Cosa hai fatto dall'ultimo daily meeting ieri?
    2.2. Cosa farai oggi?

2.3. C’è qualche problema che potrebbe far si che tu non riesca a termine il tuo compito oggi?

2.3.1. Lo ScrumMaster, finito il Daily Scrum, aiuterà chi ha un problema a risolverlo.

Sprint Planning Meeting:

1. Organizzazione meeting

1.1. Si svolge all’ inizio di ogni ciclo di “Sprint”

Durante il meeting occorre:

1.2. Individuare i compiti da svolgere.

1.3. Preparare lo Sprint Backlog definendo quanto tempo ci vorrà per svolgere i compiti individuati.

1.4. Dire quanto del lavoro assegnato sarebbe utile finire entro lo sprint

1.4.1. Product Owner + Team definiscono quanto del Product Backlog si deve fare durante lo sprint

1.4.2. Solo il Team prepara un piano per affrontare lo Sprint e lo scrive nello Sprint Backlog

Sprint Review Meeting:

1. Si svolge alla fine di ogni Sprint

2. Si effettua una review del lavoro completato e non completato

3. Si esegue una demo del lavoro completato da mostrare agli eventuali stakeholders

4. Il lavoro non completo non può essere mostrato

5. Il meeting non deve durare più di 2 ore

Sprint Retrospective:

1. Tutti i membri del Team riflettono sullo Sprint passato

2. Vengono applicati miglioramenti all’intero processo basandosi su quanto discusso

3. Si risponde a 2 domande:

3.1. Cosa è andato bene durante la scorsa fase di Sprint?

3.2. Cosa si potrebbe migliorare durante la prossima fase di Sprint?

4. Il meeting non deve durare più di 1 ora

Artefatti

Product Backlog:

1.E’ una lista mantenuta lungo tutto il processo di sviluppo ed è composta da oggetti backlog.

2.La lista è composta da tutte le feature che verranno implementate in ordine di priorità.

3.E’ un documento aperto e modificabile da tutti che contiene le stime di priorità in base al “business value” (deciso dal PO) e “development effort” (deciso dal Team). Queste stime serviranno al Product Owner per definire timeline e ordine delle feature da sviluppare.

> 3.1.Nota: a parità di business value, la feature che si sviluppa in meno tempo ha priorità (per teoria di Return of Investment). Gli indici di priorità vengono definiti con una scala da 1 a 5.

Sprint Backlog:

1. E’ la quantità di lavoro che il team svolgerà durante la prossima sessione di Sprint.
2. Le feature da sviluppare vengono divise in “task”.
3. I task sono descritti ad un livello di dettaglio accessibile da tutti i membri del gruppo, in modo che chiunque possa prendere un task dalla lista e lavorarci.
4. I task non vengono assegnati ai membri del team, ma i membri stessi hanno il compito di dividerli tra loro in base a skill e tempo: ciò promuove l’auto organizzazione e la possibilità di modificare i sottogruppi a runtime.
5. Al Backlog è sempre associata una Task Board in cui sono elencati i task e come evolverà il rispettivo stato durante lo sprint, ad es.:

1. - Aggiunta chat globale → todo
2. - Correzione tavolo da gioco → in progress
3. - Test client server su più macchine → done

Burn Down:

1.E’ un grafico che visualizza quanti task da completare rimangono nello Sprint Backlog.

2.Viene aggiornato ogni giorno e ne viene compilato uno per ogni Sprint/Release o per il progetto totale nei casi di progetti minori (Burn Down del Product Backlog)

Manuale utente

Registrazione e login

Una volta lanciata l'applicazione, la prima schermata che si presenta all'utente è quella di login. Per autenticarsi al sistema è necessario inserire username e password precedentemente registrati. Nel caso in cui l'utente non sia ancora registrato al sistema, può accedere all'interfaccia di registrazione cliccando il relativo bottone. La nuova schermata richiede l'inserimento obbligatorio di username, password, nome reale ed indirizzo email. Una volta registrato, l'utente potrà eseguire il login.

Lobby

Una volta effettuata l'autenticazione l'utente accede all'interfaccia lobby. Qui sono disponibili le liste delle stanze da gioco e degli utenti connessi al sistema ed è possibile accedere a diverse funzionalità.

Aggiorna liste

Funzione che permette di aggiornare le liste utenti e tavoli presenti in un dato istante

Creazione nuova stanza

L'utente può creare una nuova stanza cliccando l'apposito bottone. Nella nuova interfaccia occorre inserire obbligatoriamente il numero di giocatori automatici (da 0 a 3) e il nome del tavolo. L'utente che crea la stanza entra automaticamente nel gioco come primo giocatore ed ha il ruolo di "owner" del tavolo. A questo punto l'utente rimane in attesa che si colleghino i giocatori necessari a far partire il gioco.

Dettaglio stanza esistente

L'utente può visualizzare lo stato di una stanza esistente. Per farlo deve cliccare due volte sul nome della stanza stessa, presente nell'apposito elenco: una nuova finestra mostrerà in dettaglio lo stato della stanza (avviata o non avviata).

Join stanza esistente

Una volta all'interno della finestra "dettaglio stanza", l'utente potrà entrare in una stanza già creata. Nel caso in cui la stanza sia già attiva l'utente entrerà come spettatore, mentre, nel caso in cui il gioco non sia ancora avviato e vi siano ruoli liberi nel tavolo, entrerà come giocatore e gli verrà assegnata la prima postazione libera.

Dettaglio utente

Cliccando su un username presente nell'elenco della lobby, è possibile accedere al dettaglio dell'utente selezionato, visualizzando il suo stato attuale (disponibile, in attesa o occupato) e il relativo punteggio di gioco.

Chat box

Con l'ingresso nella lobby, in automatico viene aperta una finestra con uno strumento di chat che permette la comunicazione con tutti gli utenti connessi alla lobby

Logout

Cliccando su logout l'utente verrà disconnesso e l'applicativo verrà chiuso.

Tavolo di gioco

Una volta che l'utente ha creato un nuovo tavolo di gioco oppure è entrato a far parte di uno già esistente, si aprirà l'interfaccia grafica di gioco.

Gioco

Come previsto dal regolamento di Hearts, la prima fase di gioco prevede che ogni utente scelga tre carte tra quelle in suo possesso e le passi al giocatore immediatamente alla sua sinistra. Una volta che tutti e quattro i giocatori hanno terminato questa fase, inizierà il gioco vero e proprio. Ogni giocatore lancia una carta e tutti gli altri dovranno rispondere (se possibile) con una carta dello stesso seme. Ogni mano di gioco sarà vinta dal giocatore che avrà lanciato la carta con il valore più alto. Il vincitore della partita sarà invece chi avrà effettuato meno punti totali e riceverà 4 punti; tutti gli altri riceveranno -1 punto.

Esci

Cliccando sul tasto esci, il giocatore abbandonerà il tavolo facendo ritorno alla lobby. Se il tavolo è in stato "in attesa" e il giocatore è anche il creatore del tavolo, quest'ultimo verrà distrutto. Invece se è in corso una partita, allora il tavolo verrà eliminato sia se il giocatore è il creatore o no.

Problemi conosciuti

Login

Accesso negato dopo tentativo di login errato:
Quando si cerca di accedere al sistema inserendo un nome utente e/o una password non corretti, il sistema, correttamente, invita l'utente a reinserire i dati di accesso. Il seguente tentativo di accesso, anche se effettuato con credenziali corrette, non va a buon fine. Potrebbe trattarsi di un problema delle API Smack oppure del server Openfire. Gli sviluppatori hanno provato a documentarsi riguardo questo problema esponendolo presso la community *Igniterealtime*. E' possibile leggere due discussioni riguardanti questo bug ai seguenti indirizzi:

http://community.igniterealtime.org/message/216715#216715

http://community.igniterealtime.org/message/216769#216769

Chiusura tavolo

Crash su Linux e Windows:
Quando la finestra del tavolo di gioco viene chiusa (a partita finita o a partita in corso tramite il tasto esci) sui sistemi Linux e Windows viene generata un'eccezione che causa la chiusura del programma. Il problema non si presenta sui sistemi Mac. Gli sviluppatori sospettano che il bug dipenda dalla versione della JVM installata.