eChat: Facebook integrated chat using JADE on Android

SOMMARIO

OBBIETTIVO

L'obbiettivo del nostro lavoro è quello di utilizzare la piattaforma Jade al fine di sviluppare un'applicazione per smartphone Android in grado di gestire una chat orientata all'organizzazione di eventi.
L'idea di base è quella di sfruttare l'integrazione con il social Facebook per l'autenticazione dell'utente e la ricerca degli amici con cui intraprendere la conversazione; una volta all'interno della chat vera e propria saranno disponibili diverse features tra le quali la possibilità di decidere la meta dell'evento che si intende organizzare.
Il calcolo della destinazione preferita da tutti i partecipati al gruppo non è banale: ogni utente dovrà infatti rendere nota la sua posizione per abilitare il sistema al calcolo del baricentro tra tutti gli utenti partecipanti alla chat; solo una volta disponibile ad ogni partecipante il baricentro della conversazione sarà richiesta una preferenza sul luogo da raggiungere.
Terminata la fase di votazione sarà notificata ad ogni utente la meta più gradita dall'intero gruppo e sarà quindi possibile accordarsi per orario ed altri dettagli in maniera manuale tramite la chat del sistema.

JADE

Java Agent DEvelopment Framework, o JADE, è un framework sviluppato in Java che supporta lo sviluppo di applicazioni distribuite e basate sul paradigma di programmazione ad agenti, fornendo un insieme di servizi di base, conformi allo standard FIPA e necessari alla creazione e al mantenimento di un sistema multiagente.
JADE è un software open source e viene distribuito con licenza LGPL.

PIATTAFORMA

La distribuzione di JADE include un ambiente di runtime, alcune librerie che il programmatore può utilizzare per sviluppare le proprie applicazioni ed alcuni tool grafici per attività di amministrazione e monitoraggio.
Ogni istanza dell'ambiente runtime di JADE è chiamata platform (contenitore) ed al suo interno vi è un main-container ed altri container opzionali ad esso connessi. È possibile avviare più istanze di Jade e dunque più platform. All'interno di ogni platform deve sempre essere attivo il main container, il primo container ad essere attivato alla partenza della piattaforma.
All'interno del main container, in conformità a quanto previsto da FIPA, sono presenti degli agenti con ruoli speciali deputati alla gestione della piattaforma stessa ed in particolare:

- Agent Management System (AMS): è il supervisore della piattaforma, controllandone il suo accesso e il suo utilizzo. È responsabile delle operazioni di creazione e terminazione di agenti e container e dell'autenticazione e registrazione degli agenti mediante l'assegnazione ad essi di un AID (Agent IDentifier, identificatore di agente) univoco all'interno della piattaforma stessa. Fornisce, inoltre, il servizio di pagine bianche della piattaforma, mantenendo un elenco di tutti gli agenti che in un certo istante risiedono nella piattaforma stessa e memorizzando per ciascuno di essi il relativo AID.

- Directory Facilitator (DF): fornisce il servizio di pagine gialle della piattaforma, mediante cui un agente può pubblicizzare i propri servizi e/o ricercare servizi offerti da altri agenti.

AGENTI E BEHAVIOUR

Come precedentemente accennato JADE è stato sviluppato completamente in Java e, pertanto, la creazione di un agente in JADE corrisponde alla definizione di una classe che estende la classe jade.core.Agent.
Gli agenti attivi all'interno di un container JADE sono responsabili dell'esecuzione dei compiti loro assegnati.
In JADE i compiti assegnati ad un agente sono modellati mediante un'astrazione chiamata behaviour (comportamento). Il programmatore può definire specifici behaviour ed assegnarli agli agenti di una piattaforma estendendo la classe jade.core.behaviours.Behaviour.
Sono previste due tipologie di behaviour, realizzate come sottoclassi della classe behaviour: i SimpleBehaviour ed i CompositeBehaviour.

MODELLO DI COMUNICAZIONE

Uno degli aspetti chiave di un sistema multiagente è la comunicazione. Gli agenti devono, infatti, essere in grado di comunicare gli uni con gli altri, per cooperare, collaborare, negoziare e così via. Il modello di comunicazione adottato da Jade è l'Asynchronous Message Passing in cui ad ogni agente è associata una coda di messaggi ricevuti dagli altri agenti, aggiornata ogniqualvolta arrivi un nuovo messaggio. Le modalità e le tempistiche di recupero dei messaggi è demandata alla logica applicativa dei singoli agenti.
Il formato dei messaggi scambiati dagli agenti segue le specifiche del linguaggio ACL (Agent Communication Language) definito da FIPA.

ONTOLOGIE

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 che trova frequenti applicazioni nell'ambito dell'intelligenza artificiale e della rappresentazione della conoscenza.

Un'ontologia rappresenta un modo formale per definire un vocabolario comune di concetti relativi ad un preciso dominio, nonché le relazioni che intercorrono tra di essi.
JADE fornisce numerosi strumenti software per la definizione di ontologie.

WEB SERVICES

I Web Services sono ormai uno degli argomenti più importanti nel panorama attuale del software distribuito e stanno diventando una sorta di standard per l'iterazione tra le diverse applicazioni.
Avendo Jade messo a disposizione un add-on specifico chiamato WSIG, ci siamo impegnati per comprenderne il funzionamento e testarlo al fine di prendere in considerazione l'idea di un suo possibile utilizzo all'interno della nostra applicazione.

JADE WSIG

L'obbiettivo di WSIG è quello di esporre dei servizi forniti dagli agenti e pubblicati nel Jade DF come normali web services, senza particolari sforzi o configurazioni aggiuntive.
E' data allo sviluppatore notevole flessibilità di utilizzo di questo componente per venire incontro alle diverse esigente di requisiti che si possono incontrare durante un qualsiasi progetto.
L'idea è quella di generare un WSDL (Web Services Description Language) per ogni service-description registrato presso il DF e di rendere possibile la pubblicazione del servizio in un registro UDDI (Universal Description Discovery and Integration).

Questo add-on supporta tutti gli standard che riguardano il mondo dei web services:

- WSDL per la descrizione dei ws
- Protocollo SOAP per lo scambio di messaggi
- Repository UDDI per la pubblicazione

Come si può notare dalla figura WSIG è una web application composta da due elementi principali:

- WSIG Servlet
- WSIG Agent

La WSIG Servlet rappresenta il front-end con il mondo di internet ed è responsabile di:

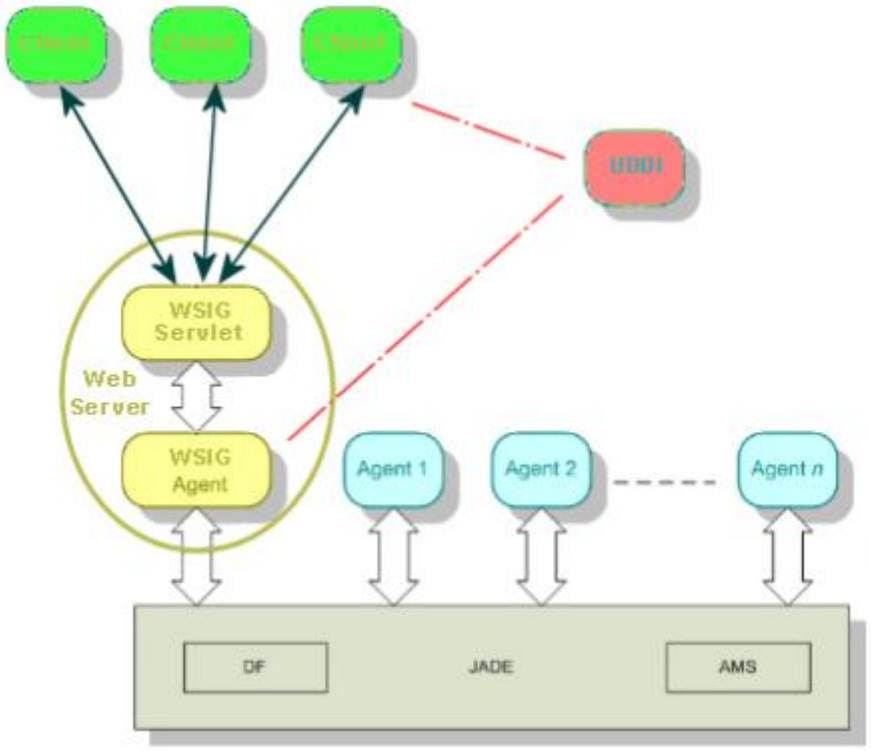
- Servire le richieste entranti http/SOAP
- Estrarre i messaggi SOAP
- Preparare la corrispondente azione dell'agente e passarla all'agente WSIG

Ed una volta che l'azione è stata servita:

- Convertire il risultato dell'azione in un messaggio SOAP
- Preparare la risposta http/SOAP ad essere inviata al client

L'agente WSIG è l'intermediario tra il web ed il mondo ad agenti ed è quindi responsabile di:

- Inoltrare le azioni degli agenti ricevute dalla Servlet agli agenti in grado di eseguirle effettivamente e di riceverne la risposta
- Registrarsi presso il DF per ricevere notifiche sull'entrata e sull'uscita degli agenti dal sistema
- Creare il WSDL corrispondente ad ogni servizio registrato presso il DF e registrarlo tramite un registro UDDI se necessario.



Ci sono due processi principali sempre in esecuzione all'interno della web application:

- Il processo responsabile di interpretare le registrazioni al DF e convertirle in WSDL, completamente svolto all'interno dell'agente WSIG
- Il processo responsabile di servire le richieste entranti e fare partire le corrispondenti azioni messe a disposizione dai vari agenti, svolto all'interno della Servlet e dell'agente WSIG.

CONSIDERAZIONI

Per quanto riguarda l'implementazione non sono state riscontrate particolari difficoltà, la documentazione è piuttosto dettagliata e con qualche esempio.
Il deploy su un server è stato decisamente più complicato del previsto: problemi di compatibilità con Apache Tomcat, errori di configurazione e mancanza di informazioni sulla rete hanno reso i test quasi impossibili.
Le difficoltà di deploy e la scarsa compatibilità delle features di WSIG con lo scopo di eChat hanno portato ad un nostro cambiamento di strategia a favore di un'architettura più tradizionale.

SERVER

L'architettura del nostro programma di chat prevede che un agente manager gestisca i partecipanti risiedendo su un server esterno raggiungibile quindi attraverso la rete.
Allo stato attuale delle cose il server è installato in una macchina locale ma potrebbe, all'occorrenza, essere portato su una qualsiasi macchina senza per questo dover cambiare configurazione o codice.
La nostra necessità è quella di 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

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

APACHE e JADE

L'installazione e la configurazione di Tomcat sono quasi completamente automatizzate e presentano inoltre diversi sistemi di integrazione con i maggiori ambienti di sviluppo Java come Eclipse o NetBeans.
Il progetto in grado di lanciare la piattaforma e l'agente manager dovrà quindi disporre delle librerie di Jade e quelle relative ad Apache, in aggiunta ai file Java relativi all'agente da mandare in esecuzione.

Le pagine JSP contengono al loro interno puro codice Java e quindi sono di facile comprensione ed implementazione.
Ecco il codice per richiamare il manager all'interno della piattaforma Jade:

```
try {


jade.core.Runtime rt;
AgentContainer ac;
rt = jade.core.Runtime.instance();

Profile p = new ProfileImpl(false);
ac = rt.createAgentContainer(p);
ChatManagerAgent chat= new ChatManagerAgent();
ac.acceptNewAgent("manager",chat);
ac.start();
chat.run();
} catch (Exception ex) {
out.println(ex);
}%>
```

ANDROID

Per questo genere di applicazione, l'integrazione con dispositivi mobili di ultima generazione è una priorità assoluta; ogni sistema di messaggistica istantanea concentra le proprie risorse sulle applicazioni mobili disponibili sui diversi sistemi operativi: Android, iOS, Windows Phone.
Jade mette a disposizione diversi strumenti per far comunicare la propria piattaforma con dispositivi Android ed è per questo motivo che è stato scelto questo sistema operativo per la realizzazione dell'app mobile di eChat.

JADE LEAP

Come conseguenza dell'introduzione di dispositivi mobili sempre connessi alla rete, le reti wired e wi-fi hanno crescente necessità di integrazione tra loro.
All'interno di questo scenario diventa sempre più importante poter sviluppare applicazioni distribuite sia all'interno della rete tradizionale sia all'interno dei nuovi dispositivi mobili.

Sfortunatamente Jade, così per come è stato concepito dai suoi ideatori, non può essere eseguito all'interno di questi dispositivi per diverse ragioni:

- L'ambiente completo di esecuzione di Jade richiede un ampio utilizzo di memoria

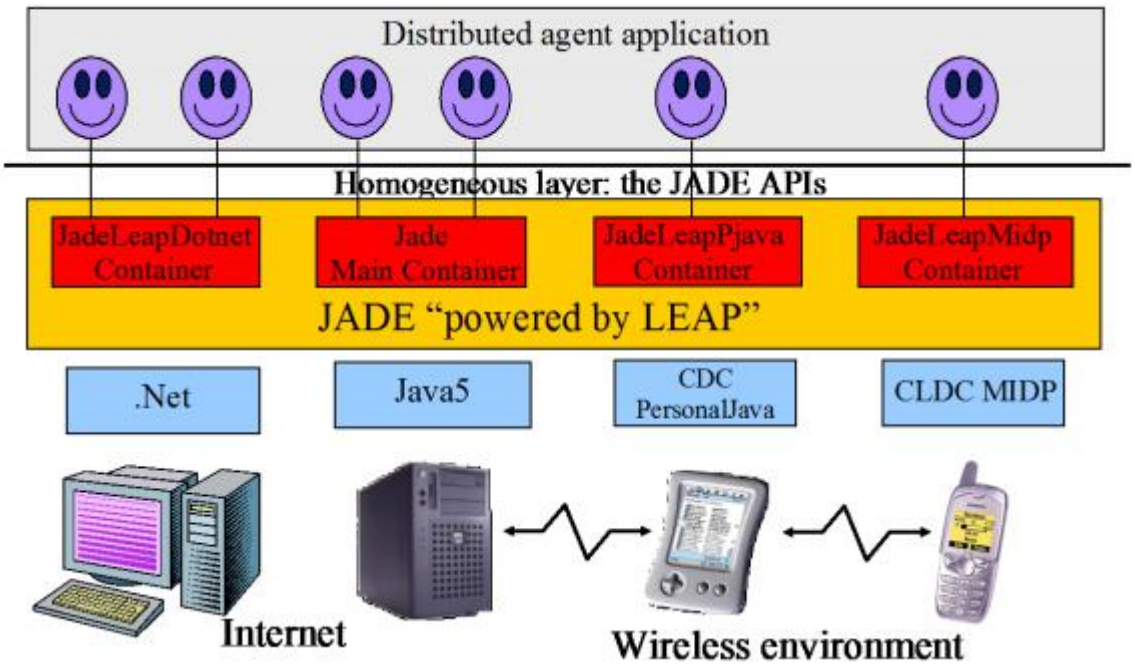
- Jade richiede una versione minima di Java non supportata da tutti i dispositivi in commercio
- Le connessioni wireless a disposizione possono presentare varie caratteristiche e problematiche come alta latenza, scarsa banda, connessione intermittente ed IP dinamico.

L'add-on Leap, quando combinato con Jade, ne sostituisce alcune parti del kernel formando un ambiente di esecuzione modificato, chiamato "JADE-LEAP" o "JADE powered by LEAP", che ha la potenzialità di essere eseguito all'interno di una vasta gamma di dispositivi:

- pjava: permette di eseguire applicazioni su dispositivi che supportano J2ME
- midp: permette di eseguire applicazioni su dispositivi che supportano MIDP1.0 (o più recenti)
- android: permette di eseguire applicazioni su dispositivi che supportano Android 2.1 (o più recenti)
- permette inoltre l'esecuzione di applicazioni su sistemi che supportano l'ambiente di runtime .NET

Jade Leap può essere modellato in diverse maniere corrispondenti alle due configurazioni (CDC e CLCD) della Java Micro Edition e dell' Android Dalvik Java Machine; fornisce inoltre esattamente le stesse API di JADE, ma con alcune differenze a runtime, che possiamo dividere in alcune macro sezioni:

- caratteristiche del JAR di JADE-LEAP e differenze relative alla command line;
- differenze implementative dell'IMTP (Internal Message Transport Protocol);
- il concetto Container Execution Mode.



JADE LEAP e ANDROID

Dopo aver descritto le motivazioni e il funzionamento generale di Jade Laep ora ci concentreremo con il suo utilizzo all'interno dei dispositivi Android.
L'implementazione di JADE-LEAP relativa ai device Android si trova all'interno del package jade.android.
Rispettando l'architettura di Android, la JADE runtime viene incapsulata in un oggetto di tipo Service.
Più in dettaglio, all'interno del pacchetto jadeAndroid.jar si possono trovare due classi:
jade.android.RuntimeService e jade.android.MicroRuntimeService, rispettivamente wrapper per un full container ed uno split container.
L'utilizzo di questi Service è esattamente uguale ad un qualsiasi altro servizio nativo di Android.
All'interno della nostra applicazione verrà utilizzato uno split container, generalmente consigliato per lavorare con dei dispositivi mobili.

Il MicroRuntimeService è dichiarato nel manifest nella seguente maniera:

```
<service android:name="jade.android.MicroRuntimeService" />
```

Il passo successivo è quello di attivare il Jade runtime attraverso l'Activity Android per fare il bind al MicroRuntimeService: una volta ottenuto l'oggetto di tipo jade.android.MicroRuntimeServiceBinder sarà possibile eseguire tutte le operazioni all'interno della piattaforma.

```
serviceConnection = new ServiceConnection() {
                                public void onServiceConnected(ComponentName className,
                                                IBinder service) {
                                        microRuntimeServiceBinder = (MicroRuntimeServiceBinder)
service;
                                        logger.log(Level.INFO, "Gateway successfully bound to
MicroRuntimeService");
                                        startContainer(nickname, profile, agentStartupCallback);
                                };

                                public void onServiceDisconnected(ComponentName className) {
                                        microRuntimeServiceBinder = null;
                                        logger.log(Level.INFO, "Gateway unbound from
MicroRuntimeService");
                                }
                        };
                        logger.log(Level.INFO, "Binding Gateway to MicroRuntimeService...");
                        this.getActivity().bindService(new
Intent(this.getActivity().getApplication(),
                                        MicroRuntimeService.class), serviceConnection,
                                        Context.BIND_AUTO_CREATE);
```

Tramite l'oggetto MicroRuntimeServiceBinder è ora possibile mettere in esecuzione un Jade Split Container tramite il codice qui presente:

```
private void startContainer(final String nickname, Properties profile,
                    final RuntimeCallback<AgentController> agentStartupCallback) {
            if (!MicroRuntime.isRunning()) {
                    microRuntimeServiceBinder.startAgentContainer(profile,
                                    new RuntimeCallback<Void>() {
                                            @Override
                                            public void onSuccess(Void thisIsNull) {
                                                    logger.log(Level.INFO, "Successfully start of
the container...");
                                                    startAgent(nickname, agentStartupCallback);
                                            }

                                            @Override
                                            public void onFailure(Throwable throwable) {
                                                    logger.log(Level.SEVERE, "Failed to start the
container...");
                                                    //---------------------------
                                                    //startAgent(nickname, agentStartupCallback);
                                            }
                                    });
            }
```

In accordo con la filosofia di Android, tutte le operazioni sono asincrone e il loro risultato è disponibile tramite un oggetto di tipo jade.android.RuntimeCallback.
Una volta sicuri della presenza del Jade runtime e del suo stato di effettiva esecuzione, sarà possibile lanciare ogni genere di agente al suo interno; in questa porzione di codice viene fatto lo start di un agente che rappresenta un partecipante alla chat:

```
microRuntimeServiceBinder.startAgent(nickname,
                            ChatClientAgent.class.getName(),
                            new Object[] { this.getActivity().getApplication() },
                            new RuntimeCallback<Void>() {
                                    @Override
                                    public void onSuccess(Void thisIsNull) {
                                            logger.log(Level.INFO, "Successfully start of the "
                                                            + ChatClientAgent.class.getName() +
"...");
                                            try {
                                                    agentStartupCallback.onSuccess(MicroRuntime
                                                                    .getAgent(nickname));
                                            } catch (ControllerException e) {
                                                                                            }
                                    }

                                    @Override
                                    public void onFailure(Throwable throwable) {
                                            logger.log(Level.SEVERE, "Failed to start the "
                                                            + ChatClientAgent.class.getName() +
"...");
                                            agentStartupCallback.onFailure(throwable);
                                    }
                            });
```

Si noti che l'oggetto ApplicationContext è passato all'agente come argomento. Questo permetterà all'agente di accedere alle API Android quanto ne avrà la necessità.

La maniera più pulita per implementare l'iterazione tra l'interfaccia grafica e l'agente è quella di esporre l' Object-to-Agent (O2A) interface, meccanismo introdotto dalla versione 4.1.1 di Jade.

Questo componente consente all'agente di esporre una o più interfacce che potranno essere viste ed invocate anche da componenti esterni.
Questa proposta è l'interfaccia O2A esposta per eChat:

```
public interface ChatClientInterface {
        public void handleSpoken(String sentece, String chatId);
        public List<String> getParticipantNames(String chatId);
        public boolean subscribeChat(String chatId);
        public void votePlace(String place, String chatId);
        public void createEvent(Location location, String chatId);
        public void sendLocation(Location location,String chatId);
        public void refuseEvent(String chatId);
        public boolean searchAgent(String idFB);
}
```

L'interfaccia dovrà poi essere registrata all'interno del setup dell'agente e ritrovata all'interno dell'Activity esterna tramite il comando:

```
                ChatClientInterface chatClientInterface = null;
        try {
                        chatClientInterface = MicroRuntime.getAgent(nickname)
                                        .getO2AInterface(ChatClientInterface.class);
                } catch (StaleProxyException e) {

                } catch (ControllerException e) {
                }
```

Lavorando con Jade, spesso potrà capitare di dovere far visualizzare nella GUI qualche informazione proveniente dall'agente. Nel caso di eChat questo si verifica, banalmente, nella visualizzazione del messaggio di testo inviato o ricevuto dall'utente.
Il metodo consigliato per ovviare a questa esigenza è quello di utilizzare il meccanismo di broadcasting messo a disposizione da Android ai suoi sviluppatori.
L'agente richiamerà un Intent specifico inviato tramite broadcast e questo verrà ricevuto dal componente android registrato per la ricezione di quella determinata azione.
Un esempio in eChat è l'invio della notifica che avvisa l'utente del termine della votazione e indica la meta scelta dal gruppo.
Si seguito il codice richiamato all'interno dell'agente:

```
        private void sendNotificationWinner(String winner, String chatId) {
                Intent broadcast = new Intent();
                broadcast.putExtra("winner", winner);
                broadcast.putExtra("chatId", chatId);
                broadcast.setAction("jade.demo.chat.NOTIFICATION_WINNER");
                logger.log(Level.INFO, "Sending broadcast " + broadcast.getAction());
                context.sendBroadcast(broadcast);
        }
```

E la versione richiamata all'interno del programma Android:

```
                IntentFilter notifyWinnerFilter = new IntentFilter();
                notifyWinnerFilter.addAction("jade.demo.chat.NOTIFICATION_WINNER");
                registerReceiver(myReceiver, notifyWinnerFilter);

…


private class MyReceiver extends BroadcastReceiver {

                @Override
                public void onReceive(Context context, Intent intent) {
                        String action = intent.getAction();
                        logger.log(Level.INFO, "Received intent " + action);
                        if..

                        else if (action.equalsIgnoreCase("jade.demo.chat.NOTIFICATION_WINNER")) {
                                String chatId = intent.getExtras().getString("chatId");
                                String winner = intent.getExtras().getString("winner");
                                showNotificationWinner(winner, chatId);
                        }

                }
        }
```

CONSIDERAZIONI

Grazie a diversi esempi e tutorial forniti dal sito di Jade e dalla community degli sviluppatori, l'implementazione di queste funzionalità non ha incontrato grossi ostacoli o difficoltà di implementazione.
Il sistema da noi proposto potrebbe sembrare simile a quello presente nel tutorial Jade ma, data la sua maggiore complessità e ricchezza di funzionalità, è stata necessaria una completa rivisitazione del programma e quindi una nuova struttura implementativa.
Nonostante ciò i costrutti di base descritti nella documentazione sono stati di fondamentale importanza per la realizzazione di eChat.

FACEBOOK

La nostra applicazione si appoggia sulla piattaforma Facebook per la realizzazione di alcune funzioni come il login, la creazione di una nuova chat o la scelta del luogo più vicino per quanto concerne la votazione. Questa scelta è stata effettuata sia perché nelle applicazioni moderne è indispensabile avere un'integrazione con un social network diffuso come Facebook sia perché questa piattaforma fornisce delle API efficaci e graficamente accattivanti, ideali per un'applicazione di questo genere.

AUTENTICAZIONE

Indispensabile per l'utilizzo di questa applicazione è il login, da parte dell'utente, al proprio account Facebook.
Al primo utilizzo di eChat verrà richiesta l'autorizzazione all'utilizzo di alcuni dati personali come nome, cognome, foto profilo o lista di amici. Questi dati risulteranno di fondamentale importanza per la realizzazione di una chat funzionante e graficamente all'avanguardia.
eChat tenterà ogni volta di collegarsi all'applicazione Facebook necessariamente installata nel dispositivo per riceverne i dati senza scomodare il login dell'utente che quindi verrà richiesto solamente in caso non ci sia nessun account attualmente collegato all'applicazione del social.

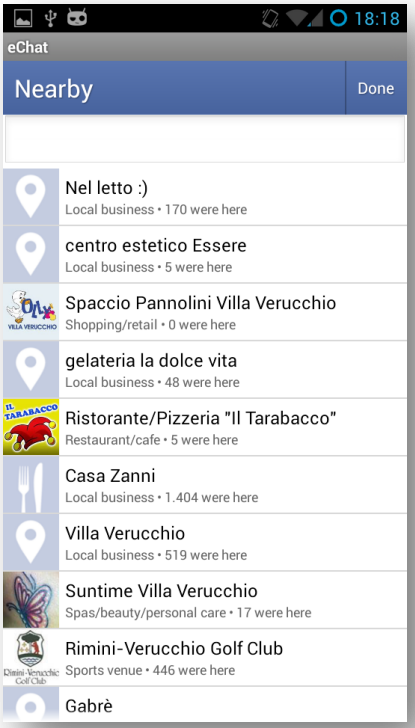
Se la fase di login non presenta problemi di alcun tipo, si potrà passare alla fase di salvataggio del dati personali e all'avvio di un agente client avente come nome l'ID Facebook appena ricevuto.

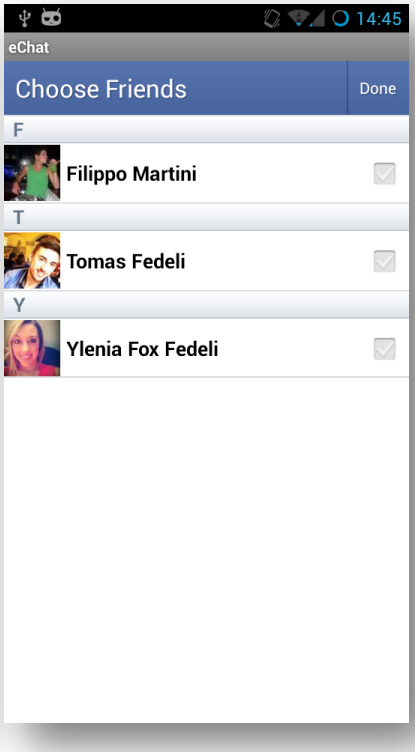
NUOVA CHAT

Per la creazione di una nuova chat è necessario, per prima cosa, sceglierne i destinatari.
Questa funzione è semplificata grazie alle API Facebook messe a nostra disposizione che forniscono un'interfaccia di selezione di tutti gli amici filtrati tramite una nostra funzione che restituisce solo quelli che hanno installato eChat nel loro dispositivo e che risultano avere un agente Jade attivo sulla piattaforma.
Una volta selezionati gli utenti con cui si ha intenzione di parlare, verrà aperta la schermata di chat vera e propria in cui compariranno le foto profilo relative ai partecipanti al fianco dei loro messaggi.

SELEZIONE LUOGO

Una volta disponibile il baricentro di conversazione, comparirà un bottone “location” con il quale sarà possibile accedere alla selezione di un’unica meta tra le tante localizzate entro un raggio di qualche kilometro dal punto centrale.
Come quella di selezione degli amici, anche questa funzionalità si appoggia alle API Facebook filtrando i risultati per categoria e vicinanza dal baricentro.





FACEBOOK E ANDROID

L’integrazione delle SDK Facebook con l’applicazione Android prevede la registrazione dell’applicazione sul sito developers del social network e diverse fasi di configurazione spiegate dettagliatamente nella documentazione messa a disposizione.
Le classi e i metodi relativi a Facebook sono disponibili tramite la dipendenza del nostro progetto eChat con un progetto-libreria scaricato dal sito ufficiale di Facebook.
L’integrazione con Jade è veramente minima parte e non ha presentato alcun tipo di incompatibilità o problema.

FAULT-TOLERANCE

Come ogni applicazione distribuita di una certa dimensione, anche eChat dispone di un suo sistema di tolleranza ai guasti.
Questo sistema consiste nella replica dell'agente manager centralizzato mediate delle features messe a disposizione dalla piattaforma Jade per aumentare l'affidabilità dell'intero programma.
Questa possibilità è piuttosto recente, infatti è stata introdotta solamente dalla versione di Jade 4.3.

VIRTUAL REPLICATED AGENT

Un Virtual Replicated Agent (VR Agent) è un agente virtuale: è solamente un nome (più precisamente un AID) che non identifica un agente concreto.
Dietro le quinte di questo nome sono presenti uno o più agenti concreti chiamati agent Replicas.
Come indicato dalla figura, Jade si occupa di dirigere i messaggi destinati al VR verso la replica scelta.
Ogni replica ha il proprio nome che differisce da quello dell'agente VR ed una propria posizione ben definita, diversamente da quanto accade per il virtual replicated agent.

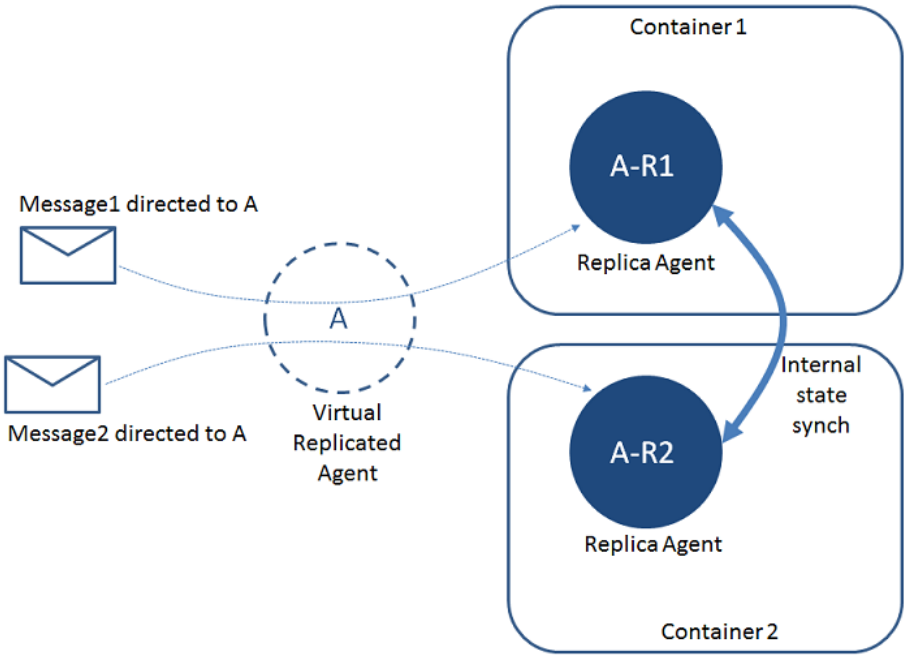


Figure 1. Virtual Replicated Agent

Gli agenti virtuali sono un potentissimo metodo per ottenere scalabilità e fault-tolerance. Sono inoltre messi a disposizione meccanismi per mantenere lo stato interno di ogni replica sincronizzato tra loro.

Nella nostra applicazione l'obbiettivo è quello di poter virtualizzare l'agente manager così da riuscire a sopperire alla possibile presenza di guasti.

Le richieste inviate al VR agent saranno in automatico ridirette verso una delle sue repliche; nel caso in cui una di queste fallisca, l'intero traffico sarà spartito tra gli agenti rimasti disponibili.
La prima replica di un agente virtuale è chiamata master replica, le altre possono essere create in qualsiasi momento e il loro stato interno è copiato da quello del master; inoltre il master replica è continuamente notificato dell'aggiunta o della rimozione delle altre repliche.
Gli agenti VR richiedono i servizi AgentReplication e AgentMobility in tutti i container della piattaforma: il primo assicura le funzionalità di replica, l'altro è utilizzato per clonare il master in caso una nuova replica debba essere creata.

Il codice che permette di mettere in atto questo meccanismo è piuttosto intuitivo e ben documentato.
Ecco un esempio di metodo da richiamare per creare una nuova replica:

```
	void createReplica(String replicaName, String where) {
		if (replicaName == null || replicaName.trim().length() == 0) {
			System.out.println("Replica name not specified");
			return;
		}
		if (where == null || where.trim().length() == 0) {
			System.out.println("Replica location not specified");
			return;
		}
		try {
			AgentReplicationHelper helper = (AgentReplicationHelper)
getHelper(AgentReplicationHelper.SERVICE_NAME);
			helper.createReplica(replicaName.trim(), new ContainerID(where.trim(),
null));
		}
		catch (Exception e) {
			System.out.println("Agent "+getLocalName()+" - Error creating replica on
container "+where);
			e.printStackTrace();
		}
	}
```

CONSIDERAZIONI

Quello appena presentato è un ottimo meccanismo per mantenere la scalabilità e garantire la fault-tolerance del programma. L'installazione e la realizzazione di questa features non presenta grosse difficoltà se non quando è necessario il deploy su Tomcat: in tal caso le configurazioni Catalina ne impediscono il corretto funzionamento.

LOGICA APPLICATIVA

Per la realizzazione di eChat è stato necessario sviluppare due macro-componenti ben distinte in grado però di interagire tra loro tramite gli agenti Jade: Il server e l'applicazione Android.

SERVER

Il server ha come unico compito quello di fare partire una piattaforma Jade e di lanciare al suo interno l'agente manager.
L'agente in questione si occupa di gestire l'iscrizione o l'abbandono da parte di un agente di una determinata sessione di chat.
Il manager memorizza le chat attive e per ognuna di esse ne gestisce i partecipanti, preoccupandosi di informare gli utenti interessati degli eventi di iscrizione o abbandono.

APP ANDROID

L'applicazione mobile ha al suo interno la maggior parte della logica applicativa.
Inizialmente verrà richiesto un login obbligatorio a Facebook, questo passaggio è fondamentale per recuperare le informazioni dell'utente connesso alla chat, poter visualizzare la sua foto profilo e soprattutto per scaricare la lista dei suoi amici.
In automatico il sistema si preoccuperà di filtrare i contatti trovati visualizzando solamente quelli che dichiarano di aver a loro volta installato eChat e di mettere in esecuzione un agente client identificato dall'ID assegnatogli da Facebook.
L'agente client svolge le sue numerose azioni interagendo con l'applicazione più strettamente legata ad Android:

- Creazione di una nuova chat singola o di gruppo
- Invio richieste di partecipazione talvolta sotto forma di notifiche
- Invio e ricezione di messaggi e relativa loro visualizzazione su dispositivo
- Creazione di un nuovo evento
- Possibilità di rilevare la propria posizione ed inviarla agli altri partecipanti della chat
- Calcolo del baricentro di chat
- Gestione di una votazione tra i membri della stessa chat per eleggere una meta comune.

La creazione di una nuova chat avviene tramite l'iterazione con l'agente manager risiedente sul Server mentre il resto della conversazione mantiene una pulita architettura P2P.
Colui che crea l'evento viene eletto automaticamente come manager e si prenderà carico di ricevere tutte le informazioni geografiche necessarie per poi calcolare il baricentro; una volta ottenuto il risultato sarà ancora suo compito distribuirlo a tutti i partecipanti del gruppo.

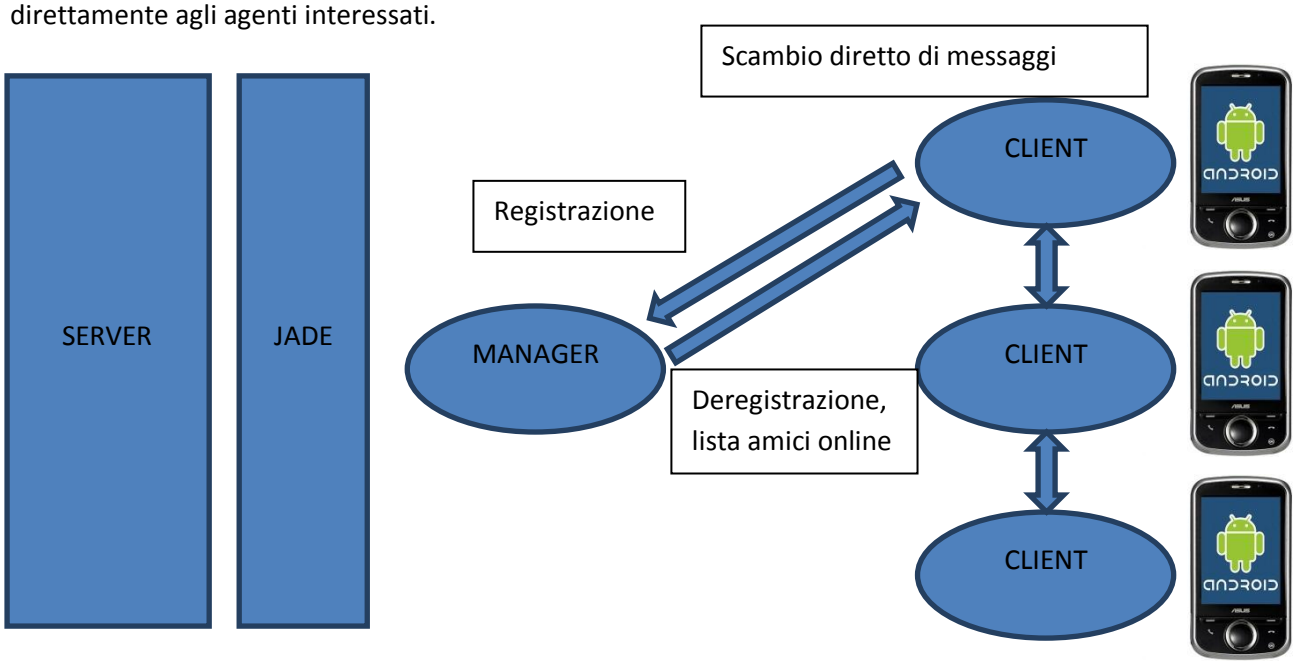
A questo punto ogni utente sarà autonomamente in grado di localizzare e visualizzare i luoghi di interesse vicini al baricentro entro un certo raggio kilometrico da impostare.
Quest'ultima feature utilizza le API formite da Facebook e consente all'utente di dare la propria preferenza.
Dopo che il manager dell'evento avrà collezionato tutti i voti, sarà quindi calcolato e divulgato il nome del vincitore.
Ogni dispositivo verrà informato tramite notifica dell'avvenuta elezione e visualizzerà a video la meta scelta dal sistema.
Per concordare orari e altri dettagli è stato deciso di lasciare agli utenti la possibilità di mettersi d'accordo tramite la chat di sistema senza ulteriori componenti aggiuntivi.

ARCHITETTURA LOGICA

Come è ovvio che sia, l'architettura di base di questa applicazione è distribuita.
E' tuttavia possibile distinguere due macro-sezioni all'interno di questo programma poiché, avendo scopi nettamente differenti, è stato necessario progettare e sviluppare due diverse architetture maggiormente adatte allo scopo.

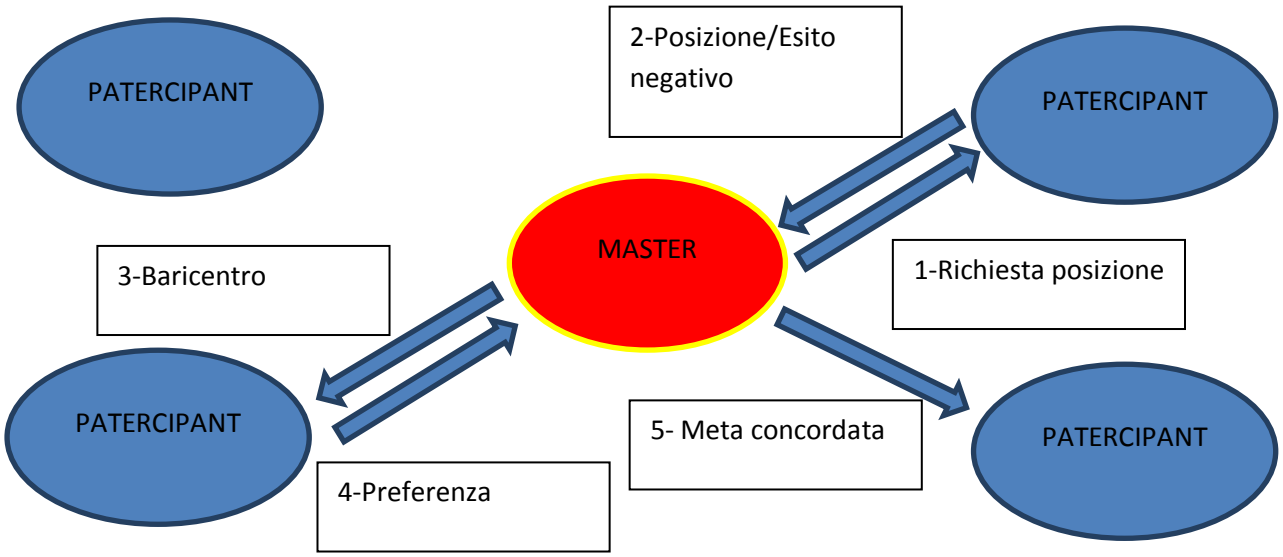
CONVERSAZIONE

Ai fini della chat nello stretto significato del termine, l'architettura si fonda su una base peer to peer con l'utilizzo di un server centralizzato solo per un'unica fase di registrazione.
Ogni dispositivo, subito dopo il login a Facebook, si registra automaticamente all'agente manager risiedente sul server Tomcat comunicando il suo Identificativo Facebook che diventerà poi il nome del suo agente associato.
Ogni agente client riceverà la lista dei propri amici attualmente online nella piattaforma e verrà notificato di eventi come registrazioni e deregistrazioni all'ambiente di chat.
Una volta avvenuta questa fase, non sarà più necessario per l'utente interagire con il server esterno.
Sarà infatti possibile creare nuove conversazioni singole o di gruppo e scambiare messaggi rivolgendosi direttamente agli agenti interessati.



CREAZIONE EVENTO

Per la creazione di un evento è necessario far parte di una sessione di Chat.
Ogni utente facente parte di un gruppo ha la possibilità di richiedere la creazione di un evento e diventarne automaticamente il master.
L'architettura, in questa fase del programma, passa da una solida base peer to peer ad una più di stampo centralizzato.
Questo passaggio è sottolineato dal fatto che venga scelto un master dell'evento che fungerà da server centrale per le restanti operazioni da svolgere, a lui infatti verranno indirizzate tutte le informazioni derivanti dagli altri partecipanti.
Il primo Passaggio della creazione dell'evento consiste nel richiedere ad ogni utente interessato la propria posizione geografica. Ogni partecipante è libero di consentire o meno al trattamento di questi dati ma invierà in qualsiasi caso al manager una risposta : una posizione geografica in caso di risposta positiva, un messaggio con un particolare contenuto per indicare un responso negativo.
Una volta collezionate tutte le risposte, il master si preoccuperà di calcolare il baricentro tra le coordinate a lui disponibili e distribuirne il risultato a tutto il gruppo.
L'ultimo compito dell'organizzatore dell'evento è quello di collezionare le preferenze dei vari utenti e di eleggerne un vincitore che sarà in seguito inviato e visualizzato da tutti.



SVILUPPI FUTURI

eChat già in sé è un'applicazione completa ma che di certo potrebbe essere migliorata con l'aggiunta di nuove funzionalità o rivisitazione di alcune sue componenti per un miglioramento delle performance.
Attualmente la chat funziona solamente con gli agenti attivi sulla piattaforma ma sarebbe di certo utile poter gestire una sorta di database con la possibilità di inviare e ricevere messaggi in modalità offline.
Il server Tomcat attualmente risiede in una singola macchina che deve pertanto necessariamente essere operativa durante lo scambio di messaggi: sarebbe interessante poter usufruire di un servizio di hosting per fare in modo di avere la piattaforma e l'agente manager Jade attivi 24 ore su 24.
Per quanto concerne l'organizzazione dell'evento, il sistema si preoccupa di automatizzare la scelta della meta (che di certo è la parte computazionalmente più gravosa) ma potrebbe, in un futuro, essere in grado di rendere automatica la scelta dell'orario, dei partecipanti e, in aggiunta, consigliare un percorso ottimale per raggiungere la destinazione.
Per quanto riguarda l'integrazione con il social Facebook, sarebbe interessante poter sfruttare in quantità maggiore le risorse date al programmatore per aggiungere funzionalità di condivisione o per analizzare una maggior quantità di dati ricavati dal social network.