UNIVERSITA' DEGLI STUDI DI PADOVA

Facoltà di Ingegneria

Corso di Laurea Magistrale in Ingegneria Informatica

TESI DI LAUREA

PARITALK: progettazione e realizzazione in java di un modulo di gestione messaggistica in PariPari

RELATORE: Chiar.mo Prof. Enoch Peserico Stecchini Negri De Salvi

CORRELATORE: Ing. Michele Bonazza

LAUREANDO: Giorgio Vallini

A.A. 2011-2012

Indice

1 Sommario **1**

2 Il Progetto PariPari **3**
2.1 I plug-in e i servizi 4
2.2 Prima degli accentratori 4
2.3 Gli accentratori 5
2.3.1 Search & Share 6
2.3.2 Profiles 7

3 Accentratore delle comunicazioni **9**
3.1 Scopo dell'accentratore 9
3.2 Interfacciamento con la GUI 9
3.3 Interfacciamento con i plugin di IM 11
3.4 Funzionalità 13
3.4.1 Gestione unificata delle location in contatti 13
3.4.2 Gestione delle conversazioni 15
3.4.3 Registro delle conversazioni 17
3.4.4 Ricerca 17
3.4.5 Notifiche 17

4 Realizzazione **19**
4.1 Principali classi 19
4.2 Workflow 20
4.2.1 Connessione con i plugin e richiesta utenti 20
4.2.2 Connessione account e ricezione dei contatti 21
4.2.3 Inizio di una conversazione 22
4.2.3.1 Conversazione in uscita 22
4.2.3.2 Conversazione in entrata 23
4.2.4 Invio, ricezione e inoltro di messaggi 23
4.2.5 Invio, ricezione e inoltro di notifiche 24
4.3 Serializzazione 25

4.3.1 Serializzazione dei contatti 25
4.3.2 Serializzazione delle conversazioni 26

5 Sviluppi futuri **29**
5.1 XMPP 29
5.2 Skype 29
5.3 Protocollo di PariPari 30

6 Conclusione **31**

Capitolo 1

Sommario

Questo elaborato conclude il lavoro da me svolto durante i due anni di partecipazione al progetto PariPari e in particolare, nel periodo da Settembre 2010 a Luglio 2012, sulla progettazione e lo sviluppo di uno dei tre accentratori previsti per la prima release pubblica del programma.

La parte iniziale (2) illustra, in modo sintetico, le principali caratteristiche della struttura del client PariPari, introducendo la nozione di plug-in. Di seguito (2.2) sono trattate le motivazioni che hanno portato allo sviluppo degli accentratori e le scelte strutturali derivanti. Segue una sezione (3) dedicata all' accentratore delle comunicazioni ove vengono presentate in dettaglio le funzionalità dello stesso; infine (4), viene presentata la realizzazione illustrata tramite diagrammi di flusso e grafici. La relazione si conclude (5) con le riflessioni sul lavoro svolto e con le previsioni in merito allo sviluppo e al miglioramento del progetto IM stesso nel prossimo futuro.

Capitolo 2

Il Progetto PariPari

Figura 2.1: Logo di PariPari

PariPari è una rete Peer-To-Peer serverless multifunzionale; il suo scopo principale è di fornire in un unico programma numerosi servizi di cui oggi si può usufruire solo utilizzando diverse applicazioni separate, in un unico programma .

Il linguaggio di programmazione usato per PariPari è Java e, nonostante presenti degli svantaggi (principalmente l'impossibilità di gestire la memoria a basso livello) molti sono i vantaggi tra i quali:

- Maggior portabilità a causa dell'indipendenza dalla piattaforma ;
- Maggior sicurezza dell'applicazione;

- Uso faciltato per l'utente finale grazie a Java Web Start che permette di scaricare ed eseguire le applicazioni direttamente dal web nonchè di avere sempre l'ultima versione del software disponibile, evitando procedure di installazione.

- Sviluppo facilitato di nuovi servizi per il programmatore grazie alle numerose API disponibili;

L'adozione dell'Extreme Programming, una metodologia agile che prevede la scrittura a più mani del codice e il suo continuo testing per renderlo robusto e privo di errori, consente di coordinare gli interventi di più persone all'interno dei gruppi di lavoro.

2.1 I plug-in e i servizi

PariPari presenta una struttura modulare molto semplice ed efficace organizzata in plug-in. Un plug-in è un programma non autonomo che interagisce con un altro programma per ampliarne le funzioni.

Grazie a questa strategia, il programma principale può essere ampliato senza necessità di essere modificato; inoltre possono assere aggiunte infinite funzioni mantenendo la medesima architettura principale. I plug-in in PariPari si dividono in due tipologie:

- cerchia interna : Core, Crediti, Connettività, Storage e DHT forniscono servizi di base per il funzionamento della rete e si coordinano fra loro per gestire richieste di risorse quali rete, disco, etc.

- cerchia esterna : Emule, Torrent, Voip, IM, IRC, DNS, etc... si basano sull'utilizzo dei plug-in della cerchia interna per fornire servizi specifici.

Il Core ha il compito di gestire le richieste ed assegnare equamente le risorse disponibili per evitare starvation[1] a favore di un solo plug-in.

2.2 Prima degli accentratori

Prima degli accentratori e della GUI (Graphical User Interface) unificata, la struttura di PariPari si presentava come illustrato in figura 2.2. La GUI "di sviluppo" era messa a disposizione dal Core e codificata all'interno di esso. Ogni plugin perciò era legato al Core sia per quanto riguarda il suo ciclo di

[1]Starvation o stallo individuale : fenomeno per il quale un processo (o un insieme di processi) monopolizza gran parte delle risorse disponibili nel sistema

vita in PariPari, sia per quanto riguarda la comunicazione con l'interfaccia grafica.

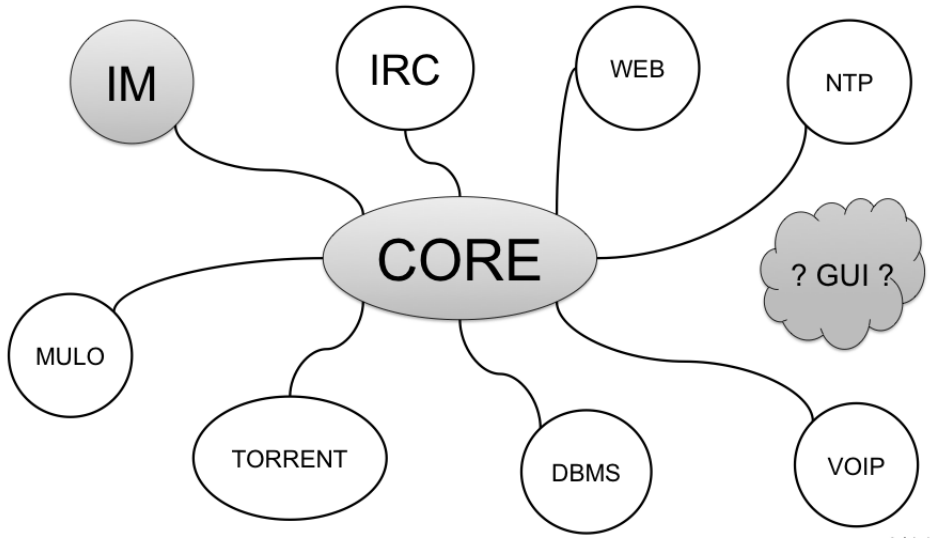

Figura 2.2: Struttura di PariPari prima dell'avvento di GUI e degli accentratori

Come si può intuire una tale struttura doveva essere migliorata. Senza una GUI centralizzata[2] era difficile coordinare i vari plugin in modo che le informazione fossero presentate in modo organizzato e user friendly. Inoltre pur avendo a disposizione una interfaccia comune, persisteva il problema della libertà di visualizzazione dei plugin riguardo la sicurezza. Senza nessun controllo infatti, un plugin malevolo potrebbe rendere inservibile il corretto utilizzo di altri plugin o dell'intero programma. Queste ed altre considerazioni riportate in seguito hanno portato al concepimento degli accentratori.

2.3 Gli accentratori

Con l'avvento della nuova GUI si è reso necessario ristrutturare l'interazione dei vari plugin con essa, in modo da poter presentare all'utente una certa organizzazione delle informazioni. In particolare, per facilitare al massimo l'ergonomia e l'usabilità, si è deciso di dividere i contenuti[3] in tre principali categorie:

[2]Per maggiorni informazioni riguardo la nuova GUI si faccia riferimento alla tesi di Laurea di Mattia Samory: PariGUI 2010.

[3]Come contenuti si intende un qualsiasi tipo di informazione (files, immagini, comunicazioni)

1. Comunicazioni (Communications)
2. Ricerca & Condivisione (Search & Share)
3. Profili (Profiles)

Per mantenere la modularità e la sicurezza, soltanto gli accentratori comunicano con la GUI, e fungono da tramite tra i plugin e la GUI.

Il loro scopo infatti è quello di accorpare e fondere i dati provenienti dai singoli plugin, organizzandoli secondo le specifiche della GUI stessa. Questa scelta sgrava i plugin dall'interfacciamento con la GUI inserendo un livello di sicurezza, che impedisce a plugin malevoli di presentare all'utente contenuti non appropriati. Il plugin infatti si interfaccia con l'accentratore seguendo protocolli specifici. In figura (2.3) si può vedere il funzionamento in modo schematico.

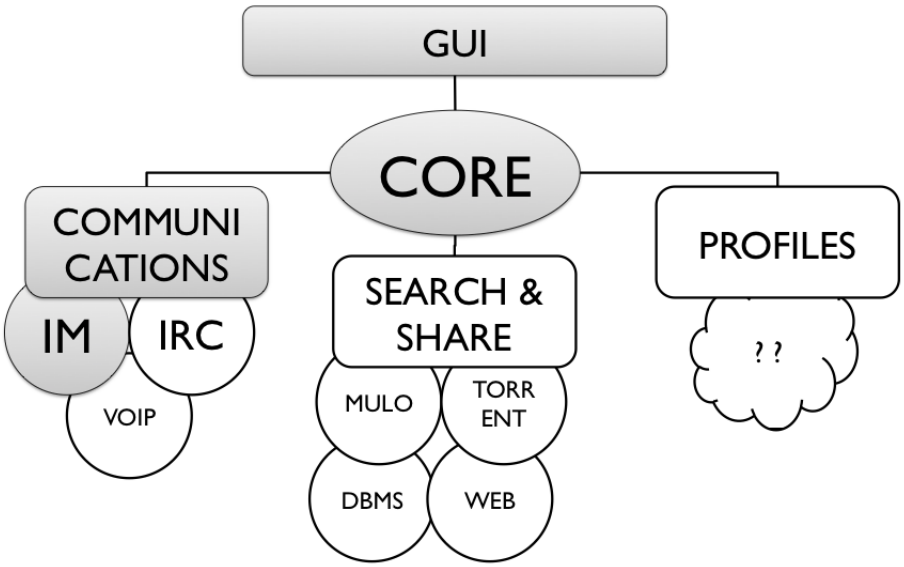

Figura 2.3: Schema di funzionamento generale tra Core, Accentratori, GUI e Plugin

Passiamo ora in veloce rassegna gli accentratori Search&Share e Profiles per poi concentrare l'attenzione sull'accentratore delle conversazioni.

2.3.1 Search & Share

Questo accentratore ha la funzione di presentare all'utente un form attraverso il quale effettuare la ricerca e la condivisione di files. L'utente potrà cercare

e condividere un file sia esso salvato nella memoria della macchina locale o in qualche rete (eMule, Torrent), in qualche Cloud Service, storage remoto o altro nodo di PariPari.

L'accentratore provvederà a inoltrare le richieste dell'utente ai plugin che offrono dei servizi compatibili ed a restituire le risposte in modo ordinato e trasparente.

2.3.2 Profiles

L'accentratore dei profili si pone l'obiettivo di unificare e presentare in forma ordinata tutti i profili dell'utente sugli attuali social network (Facebook, Google+, LinkedIn, Twitter) creando in questo modo una descrizione più completa dell'utente, ritratto nelle sue varie sfaccettature.

Capitolo 3

Accentratore delle comunicazioni

In questo capitolo sarà descritto in modo più approfondito l'accentratore delle comunicazioni, specificando le sue funzionalità ed i protocolli ideati per il suo funzionamento.

3.1 Scopo dell'accentratore

La necessità di centralizzare la gestione dei contatti e delle conversazioni ha portato alla concezione di un accentratore dedicato alla messaggistica istantanea. Esso ha principalmente la funzione di interfacciarsi con i plugin che forniscono funzionalità di messaggistica istantanea e garantire a questi la visualizzazione di contatti e conversazioni sulla GUI. Essendo tuttavia un collettore per i vari plugin, l'accentratore ha anche la possibilità di fornire funzionalità avanzate come la chat di gruppo tra contatti in diverse reti, agendo da ponte di comunicazione[1]. Ci soffermeremo in seguito sulle funzionalità avanzate dell'accentratore, mentre ora vediamo in maggior dettaglio le funzioni di interfacciamento che l'accentratore utilizza per svolgere i suoi compiti.

3.2 Interfacciamento con la GUI

L'accentratore è l'unico oggetto che comunica direttamente con la GUI; la classe Java dedicata a questo scopo prende il nome di "Bridge".

Ad alto livello il flusso d'esecuzione che l'accentratore deve gestire, può essere scatenato da queste due tipologie di eventi:

[1]Solitamente si possono effettuare conversazioni solamente tra contatti online in una stessa rete.

1. Evento scatenato dall'utente
2. Evento scatenato dalla rete

Gli eventi del primo tipo sono scatenati dall'utente di PariPari che interagisce con la GUI; l'utente potrebbe infatti decidere di:

- aggiungere, modificare o rimuovere un contatto
- iniziare, modificare o rimuovere una conversazione con un contatto o con un gruppo di contatti
- inviare messaggi in una conversazione

Tutti gli eventi di questo tipo scatenano le azioni ad alto livello esposte nella seguente tabella (colonna *Generale*); a fianco, a titolo di esempio, vediamo le azioni intraprese dall'accentratore quando l'utente che chiamiamo Bob preme il pulsante di inizio nuova conversazione nei confronti di un suo contatto, Carol.

	Generale	Esempio
1	Presa in gestione dell'evento da parte dell'accentratore	L'accentratore riceve una chiamata a BeginConversation
2	Inoltro dell'evento al/ai plugin interessato/i	Inoltro al plugin responsabile della gestione del contatto "Carol"
3	Ricezione da parte del/dei plugin di una risposta, positiva o negativa	Ricezione da parte del/dei plugin di una conferma, positiva o negativa
4	Elaborazione della risposta: se positiva viene visualizzato l'effetto desiderato sulla GUI, se negativa viene visualizzato un errore oppure si effettuano azioni correttive.	Elaborazione della risposta: se la conversazione è stata creata dal plugin viene visualizzata la finestra di chat, altrimenti viene visualizzato un errore.

Tabella 3.1: Eventi scatenati dall'utente tramite interfaccia grafica e relativo esempio

Nel caso invece di un evento del secondo tipo (non generato direttamente dall'utente ma da altri utenti nelle reti connesse) le azioni effettuate dall'accentratore sono molto più semplici:

1. Presa in gestione dell'evento da parte dell'accentratore (punto 1 della tabella (3.1))

2. Visualizzazione dell'effetto desiderato sulla GUI e eventuale inoltro ad altri plugin (punto 4 della tabella (3.1))

Dato che le azioni da intraprendere nei due casi sono l'una un sottoinsieme dell'altra, il codice è stato scritto in modo da riutilizzare le stesse funzioni (punti 1 e 4) in entrambi i flussi di esecuzione. In questo modo il flusso nel codice è lo stesso, fino alla biforcazione che distingue le due tipologie di eventi, per poi ritornare ad essere unico.

3.3 Interfacciamento con i plugin di IM

La comunicazione tra l'accentratore e i plugin è più complessa della parte vista finora; ad essa infatti è stato dedicato un intero package denominato "Comm". Ogni plugin di messaggistica, per funzionare correttamente, deve implementare le interfacce presenti in tale package e rispettare le specifiche del protocollo.

L'oggetto usato per lo scambio delle informazioni è nominato "Packet". Esso contiene:

1. informazioni generali valide in ogni stato di avanzamento del protocollo come ID progressivo, comando ;

2. informazioni particolari che attribuiscono significato al comando specifico e permettono il passaggio alla fase di avanzamento successiva.

Come illustrato in figura 3.1 la struttura dati necessaria per il trasporto dei Packet è una coda chiamata "incomingQueue" che sia i plugin che l'accentratore devono istanziare; tramite queste code l'ordine della valutazione dei pacchetti è garantito e la comunicazione è di tipo asinrono in modo da non rallentare il sistema in caso di errori.

Ogni pacchetto contiene i metodi *process* e *process_ accentrator*, il primo viene implementato dal plugin e contiene la logica lato plugin, il secondo invece viene implementato dall'accentratore e contiene la logica lato accentratore. Per garantire la sicurezza di questo meccanismo e per fare in modo che un plugin non possa sovrascrivere la logica definita dall'accentratore è stata ideata una packetFactory.

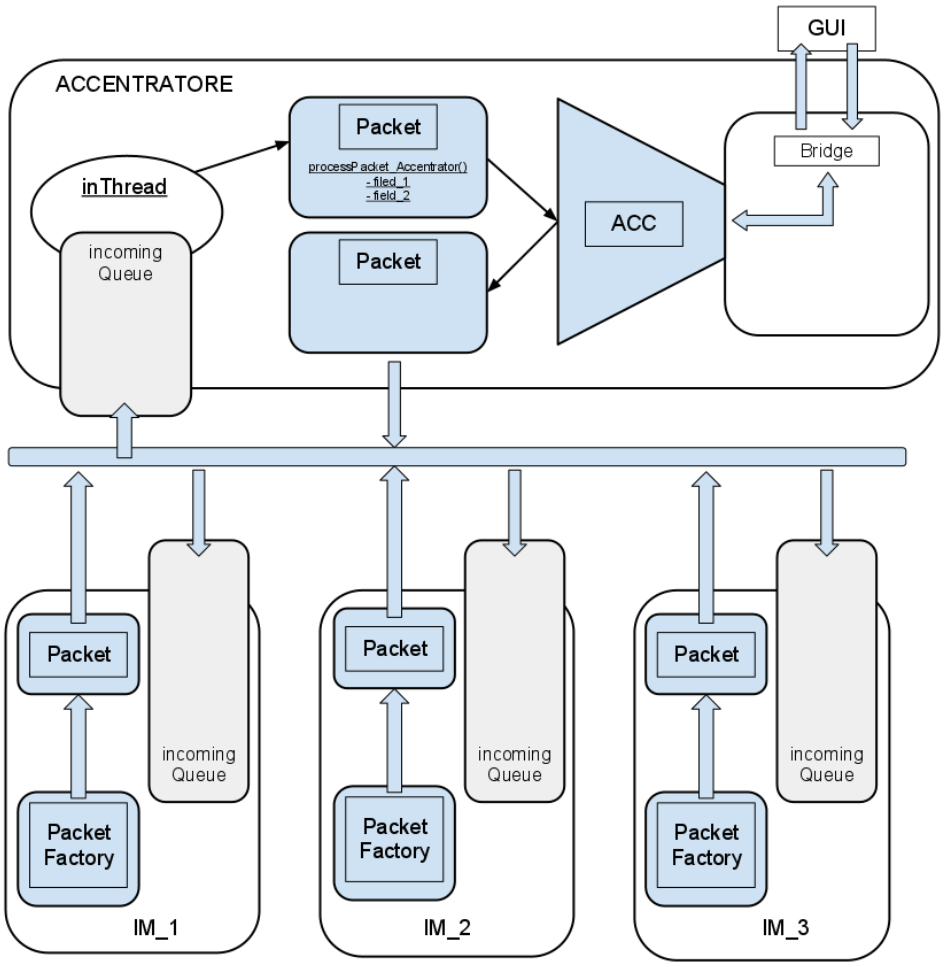

Figura 3.1: Schema di funzionamento della comunicazione tra plugin e accentratore

L'utilizzo delle factory (letteralmente fabbriche) fa parte di un pattern di programmazione chiamato Inversion Of Control[2] spesso realizzato tramite la Dependency Injection. Questa tecnica permette di disaccoppiare le logiche di servizio tramite l'iniezione di oggetti dai quali il richiedente dipen-

[2]http://it.wikipedia.org/wiki/Inversion_of_Control

de. L'utilizzo in questo caso è molto basilare, ma illustreró brevemente il funzionamento.

Le classi che definiscono il protocollo di comunicazione (package comm) sono definite nell'accentratore. Se un oggetto tra questi fosse creato dall'accentratore e poi utilizzato , non sarebbe possibile da parte del plugin la specifica del proprio codice. Per ovviare a questo problema, gli oggetti sono creati da una factory implementata dal plugin, seguendo le indicazioni di un'interfaccia nell'accentratore, ed utilizzati. In questo si ha un disaccoppiamento che permette sia all'accentratore che al plugin di implementare la propria logica di funzionamento senza intralciare quella dell'altro.

Il protocollo prevede diversi tipi di pacchetti in base alle necessità:

- RequestUsers: permette all'accentratore di conoscere quali account di messaggistica instantanea un plugin intende gestire e le relative password per effettuare l'autenticazione.
- UserAction: mette a disposizione dell'accentratore diversi metodi per agire sugli utenti ottenuti dal comando RequestUsers (create, requestContacts, connect, setStatus)
- ModifyContact: agisce sui contatti dei singoli utenti e permette di effettuare diverse operazioni su di essi (insert, delete, rename, changeBlockStatus, changeStatus)
- ConversationAction: mette a disposizione diversi metodi per controllare una conversazione (begin, addParticipant, removeParticipant, end)
- MessageAction: permette di spedire un messaggio

Una rappresentazione grafica dello scambio di questi pacchetti è riportata al paragrafo 4.2

3.4 Funzionalità

Questa sezione presenta nel dettaglio le funzionalità che l'accentratore mette a disposizione.

3.4.1 Gestione unificata delle location in contatti

La gestione dei contatti è una delle funzioni principali dell'accentratore. Il fatto di interporsi tra i singoli plugin e la GUI permette infatti di organizzare le informazioni per una visualizzazione più semplice e intuitiva.

Supponiamo di connettere con PariPari due account di messaggistica (MSN e Facebook) e supponiamo di avere tra i contatti di entrambe le reti un nostro amico (Bob).

Unificare le location in contatti significa insegnare all'accentratore che Bob nella rete MSN e Bob nella rete Facebook in realtà sono la stessa persona fisica e di conseguenza visualizzare nella GUI solamente un unico contatto.

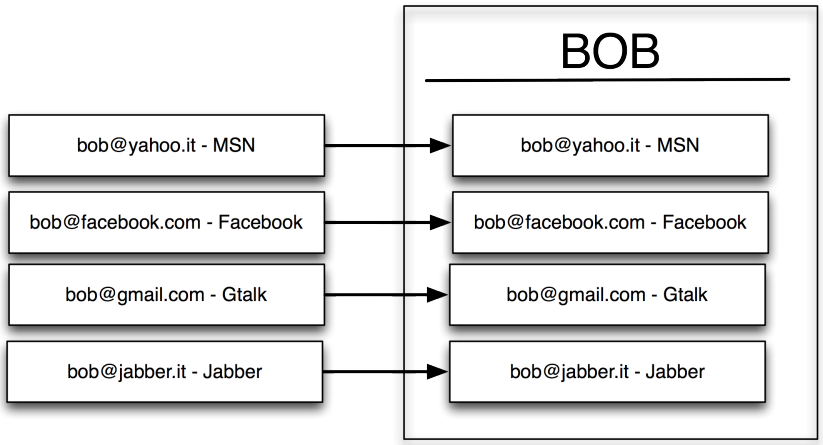

Figura 3.2: Unificazione delle location in contatti

Il contatto risulterà online se almeno una delle sue location è online, mentre l'utente potrà specificare l'ordine di preferenza delle location secondo il quale contattarlo.

L'operazione di fusione dei contatti avviene tramite trascinamento di una location su un contatto oppure tramite inserimento manuale.

L'accentratore si prenderà in carico di memorizzare tale preferenza in modo da presentare i contatti già fusi ad ogni connessione successiva.

E' importante sottolineare che questa operazione di fusione non è automatica; se lo fosse l'unico modo per inferire che due location appartengono allo stesso contatto, sarebbe effettuare dei controlli sul nickname. Questo porterebbe ad unire tutte le location con nickname uguale o simile in contatti unici, il che è un errore, che porterebbe ad unire persone diverse (ma con lo stesso nickname).

E' bene quindi lasciare che sia l'utente a decidere se e come unire le location.

3.4.2 Gestione delle conversazioni

L'accentratore gestisce conversazioni tra contatti nella stessa rete ma anche tra contatti in reti diverse.

In entrambi i casi, i messaggi ricevuti dalla GUI (quelli che scrive l'utente) vengono inviati a tutti i plugin che hanno in gestione i contatti partecipanti alla conversazione attuale (figura 3.3)

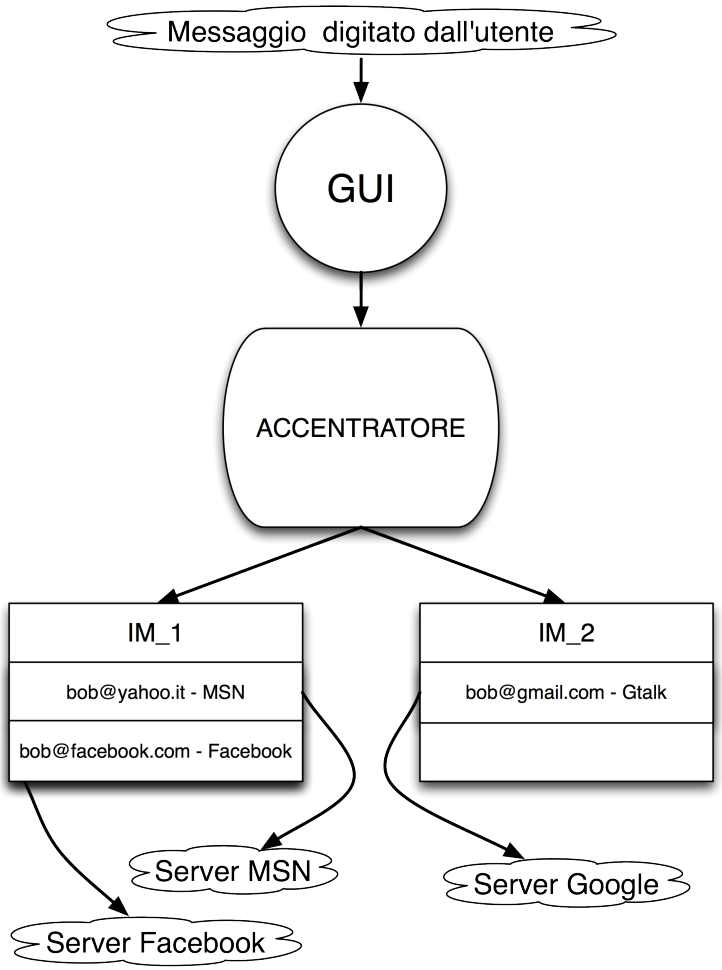

Figura 3.3: Flusso messaggi in uscita

Prima di inviare messaggi ai singoli contatti nella conversazione, l'accentratore controlla se esistono due o più contatti nella stessa rete; se così fosse e se la rete in questione supportasse le conversazioni di gruppo, allora l'accentratore comunicherebbe al plugin interessato questa necessità e sarebbe il plugin stesso a creare il gruppo.

Per quanto riguarda invece i messaggi ricevuti dalla rete

- se il destinatario è unico e quindi coincide con il contatto che ha ricevuto il messaggio, esso viene soltanto mostrato sulla GUI.

- se ci sono più destinatari oltre al contatto che ha ricevuto il messaggio, esso viene mostrato sulla GUI e inoltrato a tutti i rimanenti destinatari (vedi figura 3.4). Questo è il caso di conversazioni nelle quali i contatti non sono nella stessa rete; l'accentratore effettua un ponte permettendo a tutti i contatti di conversare come se fossero tutti nella stessa rete.

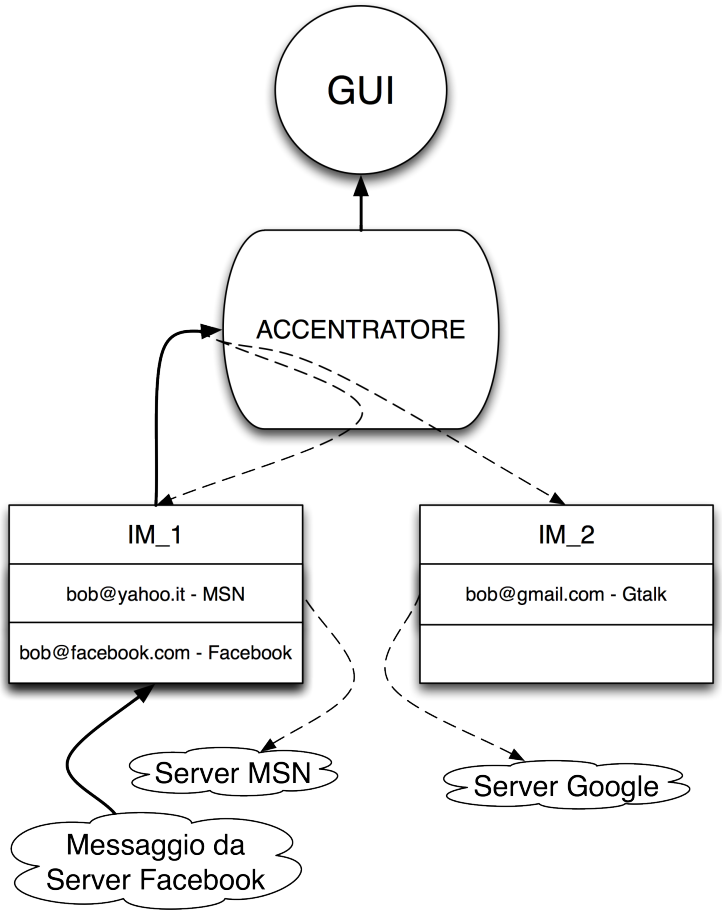

Figura 3.4: Flusso messaggi in entrata (esempio di conversazione di gruppo)

3.4.3 Registro delle conversazioni

L'accentratore, conoscendo esattamente tutto il traffico che viene scambiato tra i plugin e la GUI, è in grado di fornire dei servizi aggiuntivi, come ad esempio il registro delle conversazioni.

Ogni conversazione è serializzata su disco (vedi 4.3.2) in modo che si possa risalire in ogni momento ai partecipanti, alla data dei vari messaggi e ai messaggi stessi. Attualmente lo storico delle conversazioni non è visibile da GUI, tuttavia è una delle funzionalità che saranno implementate nel futuro.

3.4.4 Ricerca

Avendo a disposizione lo storico delle conversazioni, l'accentratore può effettuare una ricerca testuale all'interno di esse, restituendo all'utente le visualizzazioni delle conversazioni trovate che contengono il pattern ricercato.

3.4.5 Notifiche

Il sistema delle notifiche implementato da GUI permette all'accentratore di visualizzare temporaneamente a video dei messaggi.

In generale le notifiche possono avvisare l'utente di cosa sta succedendo nei vari stadi di esecuzione del programma. L'accentratore notifica:

- l'inizio/fine autenticazione account
- cambio stato contatti
- disconnessione (dovuta a interruzione della connessione oppure a disconnessione manuale)

Durante una conversazione invece:

- typing notification informano l'utente chi sta digitando un messaggio
- entrata /uscita partecipanti dalla conversazione

Capitolo 4

Realizzazione

Questo capitolo illustra gli aspetti implementativi e presenta nel dettaglio il flusso di esecuzione del protocollo specificando quali sono gli elementi coinvolti e la loro funzione.

4.1 Principali classi

Attualmente il codice dell'accentratore si trova all'interno del *Core*, più precisamente nel package *Core/centralizers/messaging*. La classe principale dell'accentratore è *MessagingCore*: in essa sono mantenute tutte le strutture dati e le funzioni di coordinamento degli altri elementi. Tra le informazioni più importanti mantenute da questa classe troviamo:

- pluginMap: lista dei plugin registrati
- conversations: lista delle conversazioni attive
- contactList: lista dei contatti
- incomingQueue: coda di ingresso per i pacchetti provenienti dai plugin
- bridge: oggetto che permette l'interazione con la GUI

Nel package *comm* sono presenti tutte le classi che implementano il protocollo di comunicazione tra l'accentratore e i plugin. Nel package GUI invece si trova la classe *bridge* a cui si è già accennato ed alcune implementazioni di oggetti come i contatti e le conversazioni.

Infine il package *utils* contiene delle funzioni di servizio come ad esempio quelle necessarie alla serializzazione su disco di contatti e conversazioni.

4.2 Workflow

In questa sezione viene presentato graficamente e spiegato nel dettaglio il flusso di esecuzione a partire dalla registrazione dei plugin fino allo svolgimento di una conversazione. Come già illustrato in precedenza gli attori in gioco sono, oltre all'accentratore, tutti i plugin che offrono servizi di messaggistica, e la GUI.

Nei successivi paragrafi si illustra il funzionamento del protocollo di comunicazione con un singolo plugin (IM) ma ovviamente il funzionamento è estendibile a molti plugin.

4.2.1 Connessione con i plugin e richiesta utenti

L'accentratore viene istanziato all'avvio di PariPari (punto 1 di figura 4.1) e come prima operazione richiede un oggetto di tipo GuiAPI; tale oggetto è necessario per comunicare con la GUI e soltanto l'accentratore lo possiede (oltre agli altri accentratori) poichè fa da tramite per tutti i plugin di messaggistica.

Successivamente (punto 2) vengono istanziati i plugin che offrono servizi di chat : in questa fase avviene lo scambio delle code di ingresso/uscita dei pacchetti tra accentratore e plugin. Più precisamente, ogni plugin, dopo essere stato istanziato, deve procedere con la registrazione tramite una chiamata al metodo *registerPlugin.* L'accentratore può così salvare il plugin associato alla sua coda, e comunicare al tempo stesso al plugin la propria coda per la comunicazione.

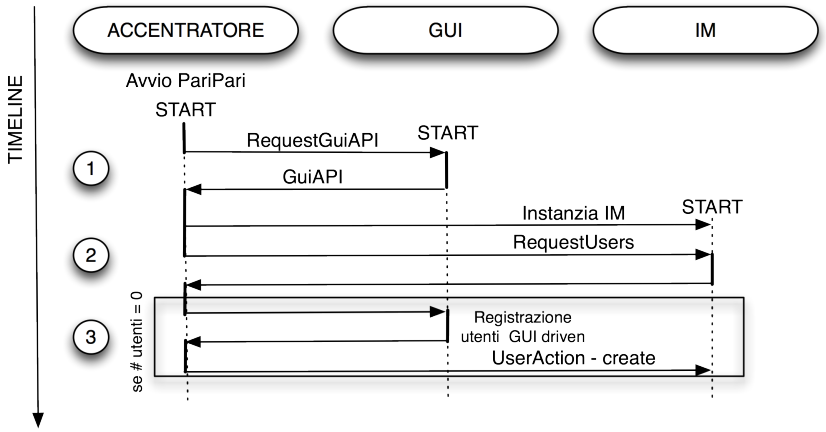

Figura 4.1: Avvio, connessione con i plugin e registrazione utenti

Infine (punto 3) vengono richiesti gli account registrati presso tali plugin (pacchetto RequestUsers). Ogni account si distingue per protocollo, email (e password). E' possibile anche registrare più account dello stesso protocollo.

Se il plugin non ha account registrati, allora l'accentratore procede con una procedura guidata che permette all'utente di inserire i propri account tramite GUI.

Al termine di questa procedura, oppure se erano già presenti account registrati, l'accentratore prosegue l'esecuzione del protocollo.

4.2.2 Connessione account e ricezione dei contatti

Per ogni plugin istanziato e per ogni utente registrato, l'accentratore inizia la procedura di login, inviando un pacchetto *UserAction* di tipo *connect* (punto 4 di figura 4.2).

Durante questa fase il plugin sottostante effettua le procedure di login ai server del caso e quando ha terminato risponde all'accentratore.

Se l'esito della connessione è positivo (punto 5) si procede con la richiesta dei contatti (location) di ogni account connesso. Una location non è altro che una coppia $< protocollo, email >$; per i plugin questo corrisponde ad un contatto dell'utente, per l'accentratore invece la location rappresenta un possibile mezzo di comunicazione per raggiungere il contatto. Questa è una fase molto delicata nella quale l'accentratore valuta ogni location ricevuta controllando se già associata a un qualche contatto oppure no. Le azioni dell'accentratore sono influenzate dalla precedente esecuzione dello stesso. Supponiamo infatti che l'utente registri il primo account in assoluto in PariPari. In questo caso non essendoci conoscenze pregresse riguardo l'aggregazione delle location, verrà creato un contatto per ogni location. Ad una successiva esecuzione però, quello che verrà mostrato all'utente sarà contemporaneamente quello che arriva dal plugin di IM, confrontato con i dati salvati le precendenti esecuzioni, che forniranno all'accentratore il modo di raggruppare le location secondo le preferenze dell'utente (punto 6).

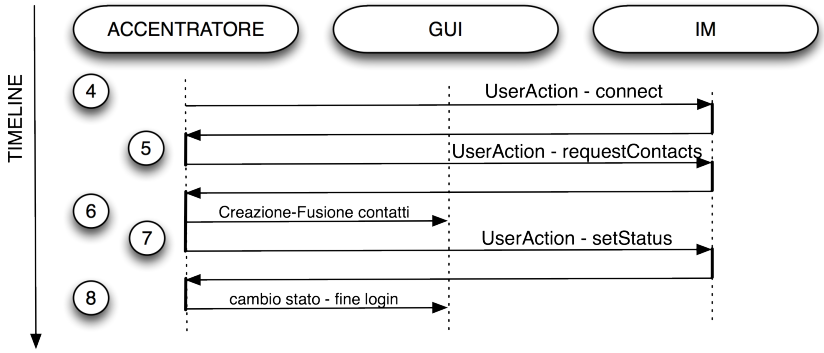

Figura 4.2: Connessione degli account, ricezione - fusione dei contatti

Per concludere, l'accentratore imposta lo stato dell'account ad *online* (punto 7) e quando la conferma dell'avvenuto cambio di stato viene ricevuta, la modifica di stato si riflette sulla GUI (punto 8) per concludere la procedura di login.

Da questo momento in poi l'utente è online e pronto a effettuare o ricevere conversazioni.

4.2.3 Inizio di una conversazione

Di seguito sono illustrate entrambe le modalità di inizio di una conversazione.

4.2.3.1 Conversazione in uscita

L'utente di PariPari intende iniziare una nuova conversazione con uno o alcuni dei sui contatti. L'evento scatenante perciò risulta essere un'azione dell'utente tramite GUI (punto 1 figura 4.3).

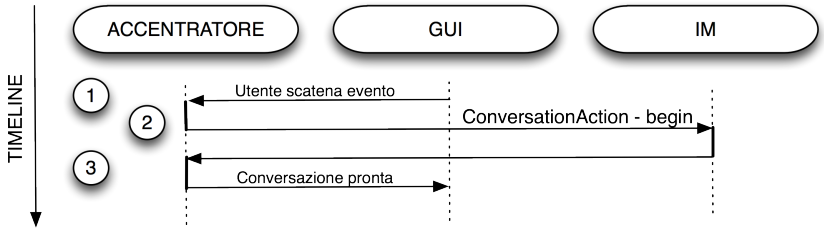

Figura 4.3: Inizio conversazione da utente

L'accentratore in questo momento raggruppa i componenti della futura conversazione per protocollo e invia separatamente richieste di inizio con-

versazione (pacchetto *ConversationAction* di tipo *begin*) a tutti i plugin che supportano tali protocolli e che hanno quelle location in gestione (punto 2).

Quando il primo plugin risponde la finestra della conversazione viene mostrata (punto 3) ; alla risposta dei plugin successivi, viene mostrata una notifica che avverte dell'entrata nella conversazione di quel contatto.

4.2.3.2 Conversazione in entrata

Nel caso delle conversazioni in entrata, l'evento scatenante arriva dalla rete, ossia da uno dei plugin registrati (punto 1 figura 4.4)

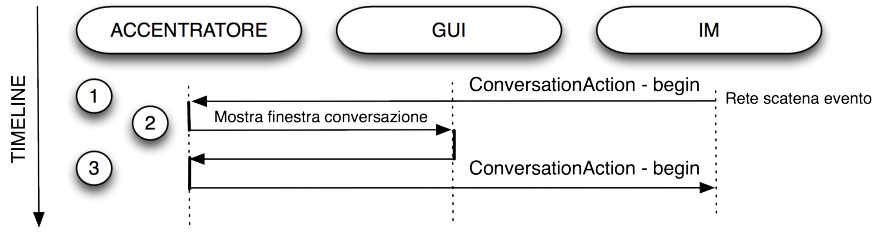

Figura 4.4: Inizio conversazione da rete

Questa situazione è leggermente diversa dal caso precedente: l'accentratore viene infatti avvertito della conversazione in entrata dopo che essa è già stata processata dal plugin. L'accentratore comunica alla GUI la necessità di mostrare una nuova finestra e riceve l'UUID che identifica la conversazione in GUI (punto 2).

Questo UUID viene comunicato al plugin (che inizialmente non lo conosceva) per le future interazioni con tale conversazione. E' cura del plugin

- mantenere con uno stato pending la conversazione in attesa dell' ID univoco che l'accentratore fornirà, e poi associare quell'ID a quella conversazione, per le future comunicazioni.

- mantenere in una coda con stato pending tutti i messaggi che potrebbero arrivare nella conversazione fino al momento in cui l'accentratore non comunica che la finestra è effettivamente aperta e pronta per ricevere messaggi.

4.2.4 Invio, ricezione e inoltro di messaggi

In modo molto simile a come vengono predisposte le comunicazioni , avviene anche lo scambio dei messaggi veri e propri. In figura 4.5 è illustrato come

vengono gestiti i messaggi digitati dall'utente, mentre in figura 4.6 è illustrato come vengono trattati i messaggi in entrata dalla rete.

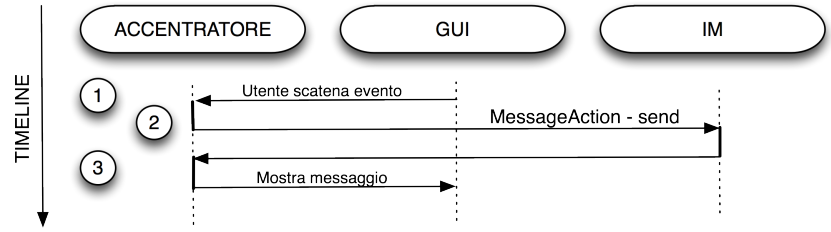

Figura 4.5: Messaggi in uscita

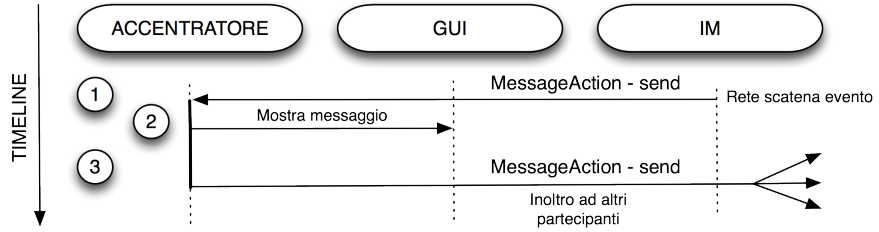

Figura 4.6: Messaggi in entrata

Nel caso di figura 3.4 si sottolinea l'importanza del punto 3, nel quale , il messaggio ricevuto da un contatto viene inoltrato a tutti gli altri contatti che partecipano alla conversazione (a prescindere dalla loro rete di appartenenza). Questa trasparenza ai protocolli rende la struttura di PariPari unica nel suo genere.

4.2.5 Invio, ricezione e inoltro di notifiche

L'accentratore ha anche il ruolo di notificare agli utenti cosa sta succedendo nella rete. Tipiche notifiche sono:

- Typing notification
- Notifiche di connessione/disconnessione contatti alla conversazione

Come nel caso dei messaggi, l'accentratore inoltra le notifiche provenienti da una sotterete, anche alle altre reti partecipanti alla conversazione, in modo che tutti i partecipanti fruiscano delle stesse informazioni riguardo lo svolgimento delle comunicazioni.

4.3 Serializzazione

Con serializzazione si intende il salvataggio dei dati su disco in un formato particolarmente semplice da codificare e decodificare : l'XML.

I dati che vengono serializzati sono

- Contatti: come detto in precedenza le location delle varie reti vengono fuse in contatti, e queste preferenze devono essere ricordate ad ogni esecuzione del programma.
- Conversazioni: per consentire il mantenimento di un diario delle conversazioni

4.3.1 Serializzazione dei contatti

L'accentratore salva le informazioni sui contatti serializzate in un file. Ogni volta che giunge una richiesta di aggiunta di una location, l'accentratore:

- se la location non è presente nel file di configurazione crea un nuovo contatto nominato con l'alias della location (se esiste) oppure con l'email della location stessa.
- se la location viene trovata nel file di configurazione significa che in qualche esecuzione precedente il contatto relativo è già stato trovato perciò legge e visualizza in GUI.

Di seguito è mostrata la struttura dell' XML per la serializzazione dei contatti e delle relative location.

```
<root>
    <contacts>
        <contact>
            <name>Giorgio</name>
            <locations>
                <location>
                    <name>email1@hotmail.it</name>
                    <protocol>MSN</protocol>
                </location>
                <location>
                    <name>email2@hotmail.it</name>
                    <protocol>MSN</protocol>
                </location>
                <location>
                    <name>email3@gmail.com</name>
                    <protocol>XMPP</protocol>
                </location>
```

```
            </locations>
        </contact>
    </contacts>
</root>
```

E' molto importante che le informazioni contenuti in questo file siano in ogni momento consistenti e coerenti con i relativi server che offrono i servizi di messaggistica. L'accentratore quindi deve propagare ogni richiesta di aggiunta, modifica o rimozione di una location al serializzatore per garantire che il tutto venga registrato.

4.3.2 Serializzazione delle conversazioni

Al contrario della serializzazione dei contatti la serializzazione delle conversazioni non è necessaria per il funzionamento dell'accentratore; è stata introdotta in un secondo momento per rendere possibile le ricerche nelle conversazioni e la compilazione di uno storico per ogni contatto.

Innanzitutto viene mantenuto un file "indice" dal quale si può capire a quante e a quali conversazioni ciascun contatto ha partecipato. L'XML ha questa struttura:

```
<root>
  <contacts>
    <contact>
      <name>Giorgio</name>
      <conversations>
        <conversation>
          <convId>722bd4d9-4477-416a-855e-4e510f3c720a</convId>
          <date>30/12/2011-14:34</date>
        </conversation>
        <conversation>
          <convId>058148de-f6c2-42a6-8556-a33b13f8ea56</convId>
          <date>30/12/2011-14:39</date>
        </conversation>
        <conversation>
          <convId>ba20a9fc-2eac-456f-b0c8-759039dde012</convId>
          <date>30/12/2011-16:19</date>
        </conversation>
        <conversation>
          <convId>601241c9-cbae-4928-96d6-a94993cf235c</convId>
          <date>30/12/2011-16:49</date>
        </conversation>
      </conversations>
    </contact>
  </contacts>
</root>
```

Il tag *name* rappresenta il nome del contatto mentre il t*ag convID* rappresenta l'UUID univoco per una conversazione.

I contenuti dalla conversazione sono serializzati in files nominati come gli UUID *convID*. Come esempio riporto una parte del file *722bd4d9-4477-416a-855e-4e510f3c720a.xml*.

```
<root>
  <convId>722bd4d9-4477-416a-855e-4e510f3c720a</convId>
  <messages>
    <message>
      <from>8a395d72-b6c7-4071-b108-eb8a401adcb6</from>
      <timestamp>30/12/2011-23:11</timestamp>
      <body> Ciao Gigi ricevi questo messaggio ? </body>
    </message>
    <message>
      <from>fb2cad21-cc35-4ccb-9ee4-71d14a084492</from>
      <timestamp>30/12/2011-23:11</timestamp>
      <body>si lo ricevo!</body>
    </message>
  </messages>
</root>
```

Come si può notare, il tag *convId* coincide con quello riportato nell'indice. I messaggi che sono stati scambiati tra i partecipanti, ognuno con il prorio mittente, timestamp e contenuto.

Seguendo questo schema di serializzazione la ricerca delle conversazioni per contatto è molto efficiente poichè il contatto è la chiave di indicizzazione. Tuttavia una ricerca testuale su tutte le conversazioni risulta meno efficiente poichè è necessario aprire tutti i file ed effettuare una ricerca in ognuno di essi. Diversi altri client di messaggistica utilizzano l'approccio indicizzato sul contatto, questo il motivo per cui mi è sembrata una buona idea anche per l'accentratore.

Capitolo 5

Sviluppi futuri

Grazie alla GUI e all'accentratore, finalmente l'utente è in grado di usare PariPari per chattare.

Un obiettivo importante è ora quello di espandere il numero di protocolli supportati dato che attualmente l'unico protocollo implementato e funzionante è MSN, sviluppato dal sottoscritto come progetto per la Laurea Triennale. Durante i successivi due anni ci sono stati dei tentativi di implementazione di XMPP che purtroppo non hanno avuto buon esito.

5.1 XMPP

La priorità è lo sviluppo di un client XMPP integrato con IM e in grado di comunicare con l'accentratore. Il protocollo XMPP ci permetterà di comunicare con client Gtalk, Facebook e Jabber. Anche Microsoft ormai da Gennaio 2012 ha reso MSN compatibile con XMPP, dato il calo di utenza subito negli ultimi anni; inoltre già da un anno i server Microsoft non supportano le vecchie versioni del protocollo MSN e sembra che a breve anche la versione 15 (quella attualmente implementata in IM) non verrà più accettata dai server.

L'opportunità che ci si pone davanti è quindi quella di poter comunicare su quattro reti diverse implementando un unico protocollo : XMPP.

5.2 Skype

L'implementazione della chat di Skype aprirebbe a PariPari una porzione grandissima di utenza. Il protocollo Skype è chiuso (ma anche quello di MSN lo è) perciò probabilmente esiste un modo per implementare un client non ufficiale.

5.3 Protocollo di PariPari

Con un client multiprotocollo che supporta le principali reti di comunicazione, la conclusione definitiva è quella di creare un protocollo utilizzabile unicamente da utenti PariPari e ovviamente in grado di fornire molti più servizi oltre la semplice chat.

Capitolo 6

Conclusione

Nel corso di questi anni ho seguito personalmente la nascita e l'evoluzione del plugin IM (per la Laurea Triennale) e la sua integrazione con l'accentratore in questa fase di transizione. Nonostante il lavoro da fare sia ancora molto per avere un prodotto fruibile dall'utenza, l'esperienza maturata nella progettazione e nello sviluppo del software è stata per me molto formativa. Oltre a questo, anche l'esperienza come team leader è stata molto importante, poichè ho avuto modo di capire come relazionarmi con gli altri partecipanti al progetto e con le persone in generale; ho avuto modo di organizzare il lavoro altrui, in modo da assegnare a ciascuno dei compiti, valutando sia le preferenze del singolo sia le necessità del momento. In conclusione questa esperienza per me non è stata soltanto un modo per acquisire capacità di programmazione e modellazione del software, ma è stata anche un'esperienza di vita che, portata avanti per tre anni, ha contribuito alla mia formazione di ingegnere.

Elenco delle figure

2.1 Logo di PariPari 3
2.2 Struttura di PariPari prima dell'avvento di GUI e degli accentratori 5
2.3 Schema di funzionamento generale tra Core, Accentratori, GUI e Plugin 6

3.1 Schema di funzionamento della comunicazione tra plugin e accentratore 12
3.2 Unificazione delle location in contatti 14
3.3 Flusso messaggi in uscita 15
3.4 Flusso messaggi in entrata (esempio di conversazione di gruppo) 16

4.1 Avvio, connessione con i plugin e registrazione utenti 20
4.2 Connessione degli account, ricezione - fusione dei contatti . . . 22
4.3 Inizio conversazione da utente 22
4.4 Inizio conversazione da rete 23
4.5 Messaggi in uscita 24
4.6 Messaggi in entrata 24

Elenco delle tabelle

3.1 Eventi scatenati dall'utente tramite interfaccia grafica e relativo esempio 10