PariGUI 2010

RELATORE: Ch.mo Prof. Enoch Peserico Stecchini Negri De Salvi
CORRELATORE: Ing. Paolo Bertasi
LAUREANDO: Mattia Samory

Corso di laurea in Ingegneria Informatica

A.A. 2009-2010

UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

TESI DI LAUREA

PariGUI 2010

RELATORE: Prof. Enoch Peserico Stecchini Negri De Salvi

CORRELATORE: Ing. Paolo Bertasi

LAUREANDO: Mattia Samory

A.A. 2009-2010

A Giovanna.

Indice

Sommario

Un prodotto software non può prescindere, oggi, da una presentazione grafica che permetta all'utente di interagire con esso in maniera diretta ed intuitiva. Ergonomia ed accessibilità sono parametri chiave nella determinazione del successo commerciale di un'applicazione.
In questo elaborato verrà descritto il lavoro svolto durante l'ultimo anno accademico per lo sviluppo di una *GUI* (*Graphical User Interface*, interfaccia utente grafica) per il progetto PariPari. Verranno trattati principalmente gli aspetti relativi all'ideazione ed al design, ponendo particolare enfasi sull'analisi dell'usabilità. Sarà infine presentato il framework realizzato per consentire l'integrazione con la piattaforma.

Introduzione

La diffusione delle connessioni a banda larga, parallelamente a quella di dispositivi portatili in grado di accedere ad Internet, ha determinato un'evoluzione nel rapporto tra uomo e computer. I servizi forniti per mezzo della rete hanno introdotto una nuova dimensione sociale al modo di pensare l'applicazione, che sta entrando in maniera pervasiva a far parte dello stile di vita quotidiano di un'ampia fascia della popolazione. Ciò ha portato alla definizione di nuovi modelli per lo sviluppo di servizi, che possano venire incontro ai possibili differenti contesti d'uso, agli obiettivi degli utenti ed al mutamento delle tecnologie a disposizione. E' emersa la necessità di formare una teoria unificata per lo studio della comunicazione e dei meccanismi d'interazione, ossia l'esigenza di affiancare all'informatica gli apporti di altre discipline, quali la psicologia, le scienze cognitive e altre ancora. Lo strumento principale che consente agli utenti di comunicare con l'applicazione è l'interfaccia grafica: è essenziale dunque che alla realizzazione preceda un'attenta analisi del design e dell'ergonomia. La piattaforma PariPari, espressione della nuova visione del software, si propone nel mercato con caratteristiche innovative. Per poter essere realmente competitiva con le applicazioni che forniscono servizi simili è indispensabile che si avvalga di un'interfaccia utente potente ed al contempo intuitiva.
Nel presente elaborato, la descrizione dello sviluppo della GUI di PariPari inizierà con un'analisi della realtà sociale e tecnologica che forma il contesto nel quale il progetto si inserisce. Il primo capitolo tratterà della posizione che l'utente assume oggi nei confronti dell'applicazione, sottolineando l'esplicita necessità di ampliare il concetto di condivisione rispetto alla concezione precedente.
Nel secondo capitolo verranno mostrate le caratteristiche della piattaforma PariPari e le problematiche nella realizzazione di un'interfaccia che derivano dalla sua struttura, attraverso uno studio comparativo con applicazioni concorrenti.
Nel terzo capitolo si affronterà la progettazione dal punto di vista del design. Verranno analizzati in primo luogo i motivi per cui molte applicazioni falliscono nel fornire interfacce facilmente comprensibili, e le soluzioni adottate. Saranno poi presentate le motivazioni alle scelte in merito allo stile grafico, ed i principi che sono alla base dei

meccanismi d'interazione, oltre alle considerazioni sull'utente finale rispetto al quale l'interfaccia sarà ottimizzata.
Nel quarto capitolo si mostrerà come sia possibile adattare l'interfaccia grafica progettata alla struttura di PariPari; saranno descritte le fasi che hanno portato dall'ideazione alla realizzazione dell'architettura per l'integrazione con il resto della piattaforma.
In appendice saranno infine riportati e commentati degli estratti di codice di riferimento per quanto esposto nell'ultimo capitolo.

Capitolo 1

Condivisione

Per comprendere chi siano gli utenti ai quali sarà rivolto PariPari, le loro esigenze, i loro desideri, è necessario analizzare il contesto sociale ed i presupposti tecnologici con i quali il progetto dovrà confrontarsi.

1.1 Presupposti sociali

Il ruolo che la società del nostro tempo attribuisce alla tecnologia dell'informazione sta subendo profondi cambiamenti. E' possibile individuare come fattore predominante di tale metamorfosi una forte democratizzazione dell'accesso alla rete. Secondo l'ITU (*International Telecommunication Union*), le persone che hanno a disposizione una connessione ad Internet nell'Unione Europea sono il 62.9% della popolazione; negli Stati Uniti il 68.7% delle case ha accesso ad una connessione a banda larga.[1] [2] Al contempo, dispositivi a basso costo vengono dotati di capacità di integrazione con servizi forniti dalla rete.
Le istituzioni hanno preso atto dell'importanza che sta assumendo questo mezzo d'interazione: in molti paesi sono in atto piani a livello nazionale per diffondere l'utilizzo di Internet ad alta velocità. La densità di abbonamenti per connessioni *broadband* è entrata a far parte degli indici statistici del benessere di uno Stato.

Come conseguenza, una nuova generazione di applicazioni, in grado di soddisfare il desiderio degli utenti di un'esperienza più ricca ed interattiva con i contenuti, si sta affermando nel mercato in enorme crescita del Web. I prodotti figli di questa evoluzione, alla quale talvolta ci si riferisce con il termine *Web 2.0*, sebbene non si distinguano

[1] http://www.itu.int/ITU-D/ict/statistics/, *Market Information and Statistics*, 2009.
[2] U.S. Census Bureau, *Current Population Survey*, Ottobre 2009.

dalla generazione precedente per sostanziali innovazioni nelle tecnologie utilizzate, sono identificabili per mezzo di alcune caratteristiche comuni [2].

John Musser e Tim O'Reilly descrivono il *Web 2.0* come:

> *" [...] un insieme di tendenze economiche, sociali e tecnologiche che formano insieme la prossima generazione di Internet - un più maturo e distinto mezzo caratterizzato dalla partecipazione degli utenti, dall'apertura e dagli effetti della rete ,,* [3]

La filosofia sulla quale si fondano è ben definita: promuovere la condivisione dell'informazione, coinvolgere l'utente nella creazione della stessa, permettere l'estensione e l'interoperabilità con servizi differenti.
Il software viene fornito come servizio, e come tale deve potersi adattare alle esigenze dei clienti, deve essere passibile di giudizio da parte loro e deve quindi poter offrire nuove funzionalità sulla base dei bisogni espressi. E' facilmente intuibile che lo sviluppo di un'applicazione, per permetterle di concorrere in un ambiente che segua tali criteri, non possa prescindere dai concetti di ergonomia ed accessibilità.

1.2 Nuovi modelli per il software

Tale dimensione sociale trova riscontro in molte delle politiche adottate dalle aziende che si sono rivelate vincenti nel settore. Viene considerata fondamentale la comprensione dell'importanza delle periferie della rete, il *long tail* nella definizione di Chris Anderson, per sfruttare il potere dei piccoli siti che formano collettivamente la maggioranza del contenuto del Web.[4] Una delle attitudini tipiche di queste nuove applicazioni è infatti quella di mettere in grado gli utenti di utilizzare in maniera intuitiva e con un approccio fai-da-te i propri strumenti. In passato, invece, si considerava preferibile la strategia di mercato di vendere ad un numero ristretto di grandi clienti.

Un altro modello di *business* tipico degli anni '90 che è necessario abbandonare per poter competere in questo nuovo contesto è il rigido controllo delle API (*Application Programming Interface*, l'interfaccia di programmazione di un'applicazione). Il monopolio di una costellazione di applicazioni e protocolli in un sistema che mira alla condivisione pone vincoli alle potenzialità della piattaforma. Ne risultano infatti

[3] Cfr.: [3]
[4] Chris Anderson, *The Long Tail*, Wired Magazine, 12 Ottobre 2006.

inficiate le capacità di adattamento alle esigenze degli utenti; ne diminuiscono le possibilità d'integrazione, che rimangono vincolate a ciò che il fornitore è in grado di offrire; più in generale sono scoraggiati i possibili contributi da parte di soggetti esterni.

Non solo l'adozione di una singola piattaforma risulta restrittiva, ma lo è anche il considerare *a priori* il singolo PC come supporto. Questo fatto è naturale per le applicazioni web, che richiedono la presenza di almeno due computer, un *web server* ed un *client*. Stanno prendendo piede servizi distribuiti, che mettono a disposizione le risorse di insiemi di computer, attraverso l'astrazione del sistema. Allo stesso modo, ma all'altro estremo dello spettro della complessità tecnologica, dispositivi portatili dalla potenza di calcolo limitata vengono dotati di capacità di integrazione avanzate. Software che possa far uso di tali diversi supporti presenta evidenti vantaggi sui concorrenti.
Un esempio che sfrutti le potenzialità dell'integrazione di supporti differenti è l'abbinata iPod/iTunes/iTunes Store di Apple Inc., in cui un dispositivo portatile si interfaccia in maniera intuitiva con i servizi forniti da server attraverso l'applicazione. Ognuno dei componenti che fa parte del sistema offre singolarmente funzionalità complete e differenti (un lettore musicale *handheld*, un'applicazione per organizzare e riprodurre file multimediali, un negozio online per musica digitale, video musicali e film), ma esprime la totalità delle proprie capacità attraverso la combinazione.

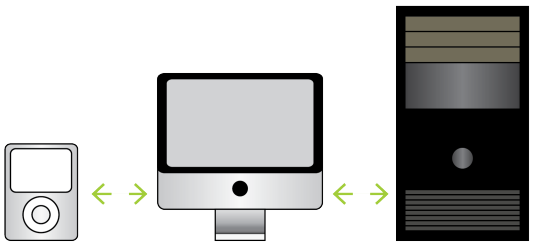

Figura 1.1: *La combinazione formata da iPod/iTunes/iTunes Store è un esempio di come possa essere sfruttata l'integrazione di dispositivi differenti*

1.3 L'utente come centro

Come il modo di pensare il software, apprendendo dalle potenzialità offerte dalla fiorente piattaforma del web, sta portando allo sviluppo di nuovi paradigmi di programmazione, così pure la modalità di interazione tra uomo ed applicazione sta cambiando rapidamente. L'attitudine delle nuove applicazioni ad usabilità ed ergonomia ha permesso una rivalutazione della posizione dell'utente, portandolo da un ruolo di semplice fruitore a quello di autore dell'informazione. Non è più sufficiente offrire contenuti statici, perché l'esperienza che ne risulta è, in confronto, limitata e sterile. E' richiesto invece un controllo del flusso dei dati su più livelli: dalla visualizzazione alla manipolazione.

Uno dei principi fondanti di questa visione è che il contributo fornito dall'utente arricchisce l'esperienza collettiva. Si può prendere come esempio la struttura adottata dagli ultimi protocolli P2P (*Peer to Peer*, rete di nodi equivalenti), dove lo sfruttamento delle periferie della rete è portato agli estremi. Ogni partecipante al *network* porta un miglioramento della qualità del servizio, avendo perso le limitazioni derivanti dall'assunzione di architetture *client-server*. Ogni *peer*, seppur perseguendo i propri obiettivi in maniera "egoistica", arricchisce la comunità con parte delle proprie risorse per il solo fatto di esserne membro. I servizi che sfruttano a pieno questi effetti della rete sono caratterizzati da una grande scalabilità e, se trovano supporto nel mercato, sono in grado di autoalimentare i propri contenuti.

Il valore aggiunto che l'utente apporta è il fondamento della ricchezza di un'applicazione.

Il rilievo dato allo *UGC* (*User Generated Content*, contenuto generato dagli utenti) ha determinato la popolarità di servizi web come Youtube o Flickr.
Il sistema di *feedback* di eBay, in concomitanza alle dimensioni del suo bacino di utenza, lo rendono sostanzialmente immune dalla concorrenza.
L'enciclopedia online collaborativa Wikipedia al momento della stesura di questo testo è al settimo posto della classifica dei siti web più visitati stilata da Alexa.[5]
La "saggezza degli utenti" decide ciò che è rilevante e cosa non lo è, compito riservato precedentemente a media considerati autorevoli: basti pensare al fenomeno del *social bookmarking*, dove gli utenti partecipano alla costruzione di elenchi di segnalibri

[5] http://www.alexa.com/topsites, *The top 500 sites on the web*.

(*bookmark*) di risorse di Internet ritenute di interesse, categorizzati secondo strutture libere e legate alla semantica di parole chiavi, per associazioni informali.

1.4 Condivisione dell'informazione

L' architettura della partecipazione assunta come *design pattern* si presenta anche sotto una diversa forma. Grazie a determinate scelte progettuali derivate dalla visione del software come un servizio (*Saas, Software as a Service*), l'autorialità trova come mezzo espressivo non solo la creazione dell'informazione, ma anche (se non principalmente) la sua rielaborazione.

Per permettere questo si è rivelato necessario adottare approcci innovativi:

Utilizzare standard di condivisione dell'informazione mirati alla syndication.

Iniziative recenti mirano alla definizione di standard per la trasmissione dell'informazione, come SOAP (*Simple Object Access Protocol*) e WDSL (*Web Service Description Language*). I più leggeri di questi sono stati ampiamente inclusi nei servizi di maggior successo.

Il sistema *RSS* (*Really Simple Syndication*), in particolare, ha portato alla rivoluzione dei blog, e non solo. Il concetto introdotto da questa tecnologia è quello di visione della pagina web non come un documento al quale vengono apportate modifiche, ma come un aggregato di contenuti in continuo mutamento, dei quali è possibile monitorare l'evoluzione. Come conseguenza, l'informazione non è più legata alla pagina, e può essere acquisita e riprodotta attraverso canali differenti.

Adottare modelli di programmazione leggeri.

La struttura tipica delle applicazioni desktop è pensata per consentire esclusivamente abbinamenti rigidi. La mentalità su cui si basano i nuovi servizi è notevolmente diversa. Adottando modelli di programmazione leggeri si permette agli utenti di rinnovare la potenza dei singoli servizi attraverso la loro combinazione in ibridi, o *mashup*.

Un secondo vantaggio è la possibilità di poter velocizzare i tempi di rilascio del software. La stessa idea di rilascio è in alcuni casi stata superata e sostituita da quella di servizio in continuo sviluppo. In questo modo si accorciano le distanze

tra sviluppatori ed utenti, e si rende più dinamica l'interazione.

Grazie all'adozione di modelli di programmazione leggeri, congiuntamente alla pubblicazione di API libere, l'utente può prendere parte attiva alla produzione dei servizi. *Mass collaboration*, concetto nato nell'ambito del software libero, è la manifestazione del nuovo atteggiamento della società nei confronti della creazione dell'informazione.

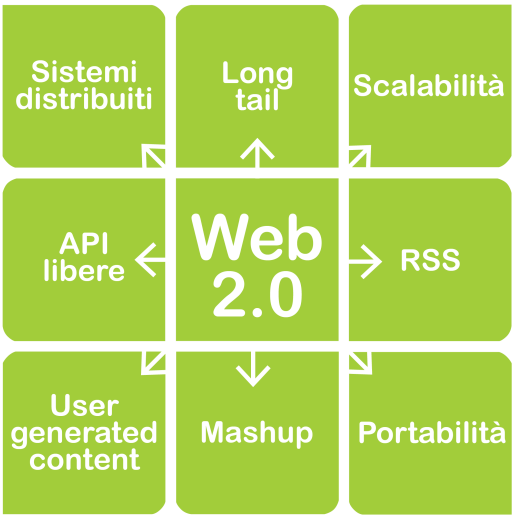

Figura 1.2: *Mappa concettuale delle caratteristiche principali del "Web 2.0"*

Capitolo 2

PariGUI

In luce del contesto delineato nel precedente capitolo, e delle esigenze di mercato che ne derivano, di seguito verranno presentati il progetto PariPari e le problematiche che questo pone nell'ideazione di un'interfaccia, in relazione alle applicazioni concorrenti più diffuse.

2.1 Le richieste del mercato

E' nello scenario creato dalla nuova visione del software come servizio che nasce la piattaforma PariPari [1]. Come è stato esposto nella sezione 1.4, l'ambiente richiede come caratteristiche chiave:

- *la capacità di promuovere la partecipazione gli utenti, coinvolgendoli nella creazione dell'informazione.* Sebbene l'intelligenza collettiva sia uno dei maggiori benefici derivanti dalla struttura della rete, per poterla sfruttare è indispensabile fornire supporti che ne facilitino la messa in atto. *In primis* è fondamentale offrire un'interfaccia grafica intuitiva, attraverso la quale anche gli utenti meno esperti possano avere un'esperienza positiva e vengano invogliati all'utilizzo del servizio, senza risultare frustrati o alienati dalla necessità di comprenderne i meccanismi.

- *il supportare estensioni e ricombinazioni delle proprie funzionalità.* Strutture monolitiche e controllo del software proprietario tramite l'utilizzo delle API sono politiche sempre più anacronistiche, nell'epoca del *Web 2.0*. Per rendere possibile la generazione di ibridi tra servizi è necessario far uso di modelli di programmazione leggeri, adottare strutture modulari e altamente flessibili.

- *il superamento dei limiti imposti dal singolo dispositivo.* Da una parte questo implica garantire portabilità, allargando l'orizzonte del proprio mercato a dispositivi con potenze di calcolo e capacità di interazione con l'utente differenti; dall'altra significa trascendere i vincoli della singola macchina, per mezzo di tecnologie distribuite.

2.2 PariPari

2.2.1 Multifunzionalità intelligente

PariPari è una rete *peer-to-peer*, basata su una variante di Kademlia, che si è posta l'ambizioso obiettivo di presentarsi come piattaforma multifunzionale.[1]
Il suo scopo principale è fornire uno strumento capace di distribuire per mezzo della rete sia le funzioni comunemente associate al mondo del *peer-to-peer*, come *file sharing*, *distributed storage* e *distributed backup*, sia i servizi tipici di Internet, quali *DNS* (*Domain Name System*), *web server*, *DBMS* (*DataBase Management System*), *IM* (*Instant Messaging*), *IRC* (*Internet Relay Chat*), *VoIP* (*Voice Over IP*).

A tal fine è stato realizzato un meccanismo di amministrazione delle risorse interne all'applicazione che rende possibile la coesistenza proattiva di tali funzionalità. E' certamente questo l'aspetto più peculiare del progetto, in grado anche di renderlo competitivo rispetto alle alternative attualmente presenti sul mercato, legate strettamente al singolo servizio.

2.2.2 Struttura flessibile

Il client adotta un'architettura modulare. L'applicazione evolve a partire da un nucleo centrale, chiamato `Core`, gestore delle risorse interne, al quale possono essere aggiunti componenti, chiamati `Plugin`, secondo meccanismi di estensione ben definiti. Un insieme di `Plugin`, la cosiddetta "cerchia interna", offre le funzionalità di base, come l'accesso alla rete o il controllo del file system: a loro si possono rivolgere i `Plugin` della periferia di PariPari per ottenere gli strumenti sui quali basare i propri servizi. Con questo approccio, che Larry Wall ha definito "a cipolla", chiunque può aggiungere `Plugin` che apportino qualsiasi funzionalità, purché vengano seguite le specifiche.

[1] http://xlattice.sourceforge.net/components/protocol/kademlia/specs.html *Kademlia: A Design Specification*.

Per creare il client è stato deciso di utilizzare come linguaggio di programmazione Java^{TM}. A fronte di trascurabili cali di prestazione, in particolare nel caso di operazioni di crittografia, questa scelta offre il vantaggio principale di poter distribuire PariPari su un gran numero di dispositivi. Java rende possibile la portabilità del programma potenzialmente su tutte le piattaforme senza bisogno di ri-compilare - come reclamizzato dal suo slogan: *"write once, run everywhere"*, ovvero "scrivi una volta, esegui ovunque".

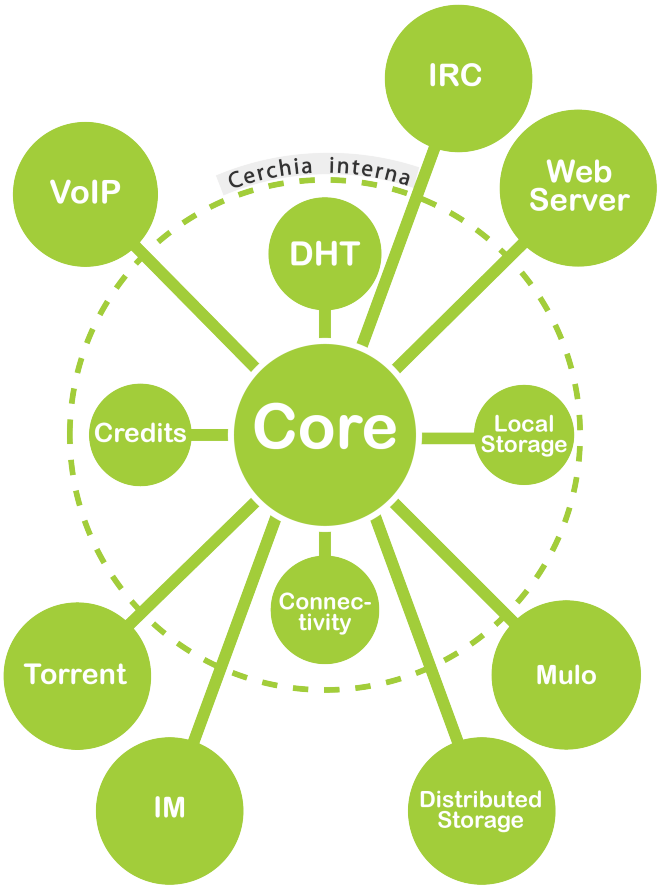

Figura 2.1: *I plug-in di PariPari sono moduli intercambiabili dinamicamente, connessi attraverso il Core. Gli appartenenti alla "cerchia interna" forniscono le funzionalità sulle quali basare le estensioni dei servizi*

2.2.3 Rete aperta

Così come vengono applicate politiche per garantire la concorrenza tra le funzionalità nel singolo nodo, il sistema di crediti permette di promuovere la partecipazione dei

nodi alla rete, favorendo quelli che condividono di più. A confronto con sistemi simili, ad esempio eDonkey2000, quello adottato manifesta una maggiore transitività nella distribuzione dei ”meriti storici” acquisiti dai singoli partecipanti.[2]

La struttura della rete di PariPari è completamente decentralizzata: questo le conferisce una notevole robustezza rispetto a possibili attacchi esterni e malfunzionamenti dei singoli nodi. Per permettere la localizzazione di *host* e risorse mantenendo l'approccio *serverless*, PariPari fa uso di una tabella di hash distribuita, o *DHT* (*Distributed Hash Table*). Un'architettura di questo tipo è caratterizzata da un'ottima scalabilità.
Una particolare attenzione è stata riposta nel garantire l'anonimato degli utenti.

Una caratteristica innovativa e al tempo stesso in linea con la filosofia originaria di Internet è l'apertura verso l'esterno della rete. I servizi offerti potranno infatti essere fruiti anche da nodi non facenti parte di PariPari.
Coerentemente con il principio di condivisione su cui si fonda, il progetto sarà *open source*, rilasciato sotto licenza GPL (*GNU General Public License*).[3]

2.3 eMule, BitTorrent, Skype, Gmail

Per comprendere cosa gli utenti, avvicinandosi per la prima volta a PariPari, si aspettino dalla sua interfaccia, è necessario analizzare le applicazioni più diffuse che possano essere prese come termine di paragone.
In questo modo è possibile individuare le strutture più comuni per la presentazione degli elementi chiave ed i paradigmi d'azione tipici ai quali l'utente è abituato.
Nell'ambito del *file sharing*, tra le applicazioni più note spiccano i client eMule e BitTorrent.[4] [5] Per la comunicazione, la popolarità di Skype e Gmail li rende modelli immediatamente riconoscibili.[6] [7]

eMule L'interfaccia grafica di eMule non si contraddistingue per intuitività. Ottimizzata per una visualizzazione a schermo intero, l'area è divisa verticalmente in tre fasce. Dall'alto in basso sono presenti: una barra degli strumenti, composta da pulsanti

[2] Cfr. http://en.wikipedia.org/wiki/EDonkey2000
[3] Cfr.: http://www.gnu.org/licenses/gpl.html
[4] Cfr.: www.bittorrent.com
[5] Cfr.: www.emule-project.net/
[6] Cfr.: http://www.skype.com/intl/it/home
[7] Cfr.: http://mail.google.com/mail/help/intl/it/about.html

ad icona; un pannello centrale, che occupa la quasi totalità della superficie visibile, dedicato alle funzionalità vere e proprie; un *footer*, dell'altezza di una riga di testo, per notifiche ed informazioni sullo stato.
Nella barra degli strumenti si incontrano da sinistra a destra tre gruppi di bottoni: il primo contiene comandi diretti di utilizzo frequente ("Connetti"); il secondo permette di mostrare nel pannello centrale differenti *viste*, ossia informazioni e funzionalità pertinenti ad una particolare categoria ("Kad", "Server", "Trasferimenti", "Cerca", "File Condivisi", "Messaggi", "IRC", "Statistiche"); il terzo mette a disposizione gli strumenti di configurazione e supporto ("Opzioni", "Strumenti", "Aiuto").
Sebbene questa disomogeneità di significati, unita alla numerosità delle opzioni disponibili, risulti di difficile interiorizzazione, offre il vantaggio di un accesso rapido alle aree del programma ove svolgere compiti specifici.
Ognuno dei pannelli delle differenti viste è organizzato autonomamente dagli altri, per essere sfruttato nella sua totalità. Per quasi tutti i pannelli, però, la maggior parte dell'area è dedicata alla visualizzazione di voluminose tabelle, contenenti informazioni dettagliate relative ai nodi con i quali si è in contatto, ai server disponibili, ai file in trasferimento ed in condivisione, ai risultati delle ricerche.
Per compiere qualunque azione l'utente ha a disposizione uno strumento dedicato, il che rende necessaria una certa familiarità con l'interfaccia.

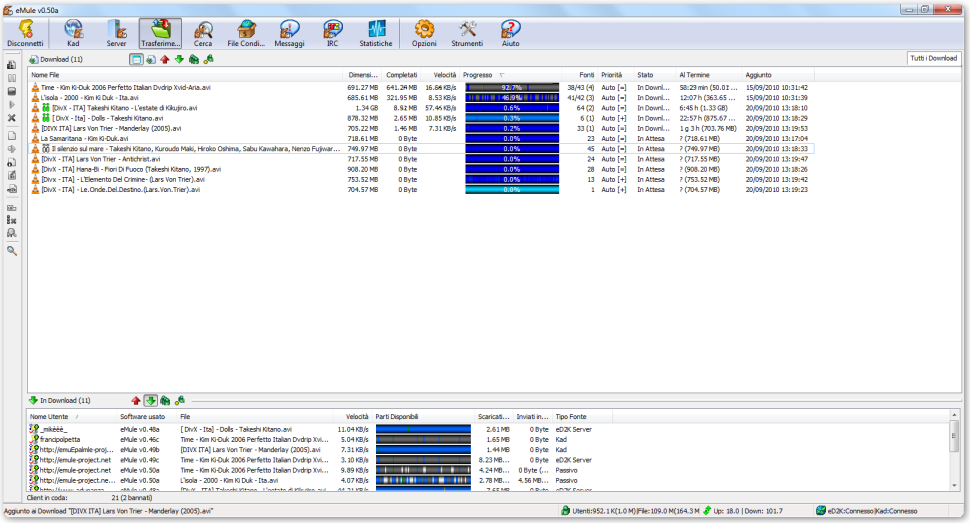

Figura 2.2: *Interfaccia grafica di eMule, versione 0.50a*

BitTorrent Nonostante molte delle funzionalità offerte siano simili a quelle di eMule, BitTorrent si presenta in maniera più compatta, minimale. Questo è dovuto alla scelta di privilegiare la vista relativa al trasferimento dei file, mostrando solo le informazioni

più rilevanti e gli strumenti di interazione d'uso più frequente. Lo scheletro dell'intrefaccia si rifà a pattern noti e di comprovata efficacia comunicativa.
Per mezzo di una barra-menù è possibile configurare il set di componenti visibili ed il comportamento del programma.
Una barra degli strumenti fornisce pulsanti ad icona per effettuare operazioni sui file in trasferimento.
L'area sottostante è tripartita secondo il modello *organizer-overview-detail*: a sinistra vi sono dei filtri per selezionare file con determinate caratteristiche; a destra, l'area dedicata alla rappresentazione dei file contiene una tabella che ne riporta le caratteristiche salienti; in basso, dei pannelli sovrapposti mostrano dettagli e statistiche.
A piè di pagina si trova una barra che riassume lo stato dell'applicazione.

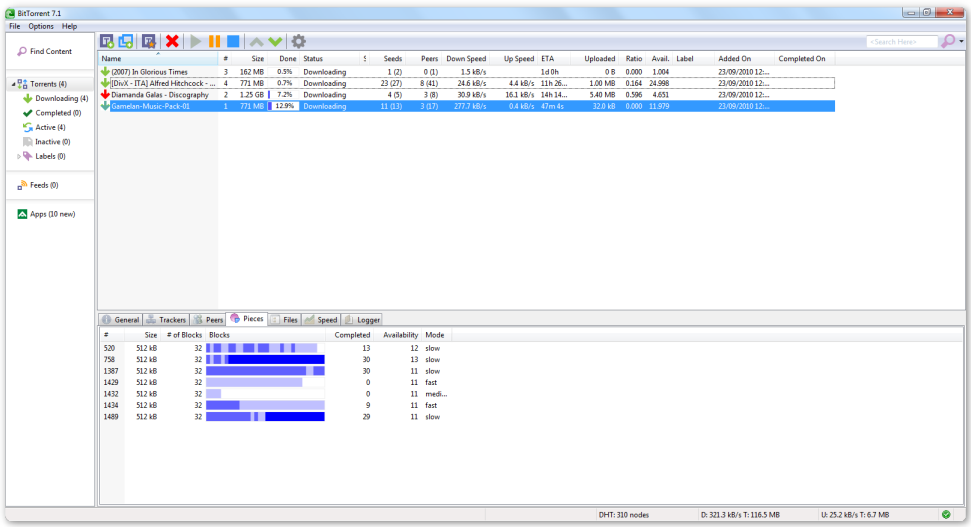

Figura 2.3: *Interfaccia grafica di BitTorrent, versione 7.1*

Skype Skype ha diviso l'area disponibile asimmetricamente, in quadranti di dimensione variabile.
Nei due pannelli della riga superiore sono riportate le sinossi dei profili dell'utente e dei partecipanti alla conversazione.
Il pannello in basso a sinistra è costituito da un *tabbed panel*, con le indicizzazioni per contatti e gruppi di conversazione.
Il quadrante rimanente è dedicato all'azione che l'utente sta compiendo: all'occorrenza, l'area di testo per la visualizzazione della chat si ridimensiona per lasciare spazio all'immagine della videochiamata.
La grafica assume un ruolo che trascende la semplice illustrazione: diventa uno strumento d'interazione, sul quale è possibile operare direttamente. Per agire ad esempio

sull'immagine di profilo è sufficiente selezionarla, lo stesso vale per il messaggio di stato personalizzato.
I dialoghi di configurazione e le funzionalità di utilizzo poco frequente sono accessibili da un menù collocato nella zona superiore della schermata.

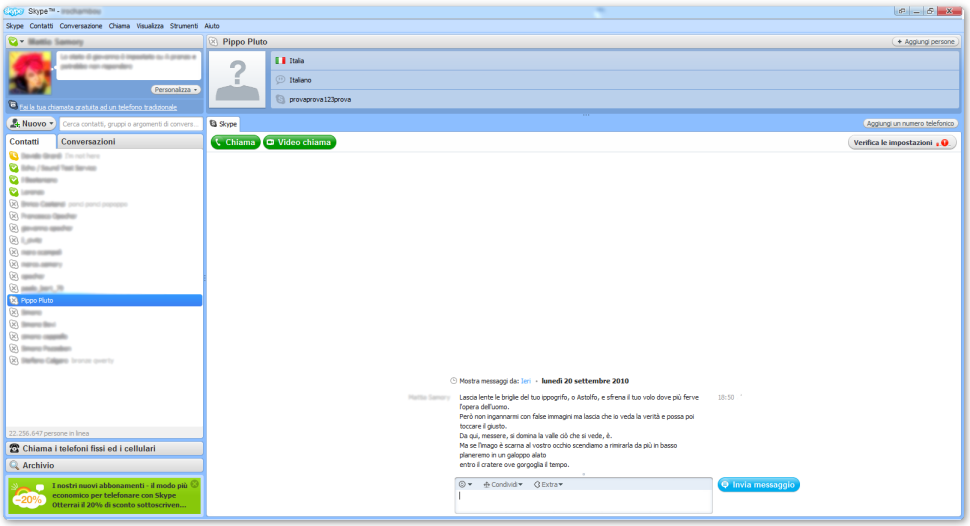

Figura 2.4: *Interfaccia grafica della versione 4.2.0.169 di Skype*

Gmail In linea con lo stile che caratterizza i prodotti di Google Inc., Gmail ha un'apparenza discreta, sobria.
La pagina è divisa in due colonne. L'area di destra permette la rappresentazione e la manipolazione dei dati corrispondenti a diverse viste. Quella di sinistra è divisa verticalmente in più segmenti: un primo consente la selezione delle viste (”posta”, ”contatti” ed ”attività”); un secondo contiene filtri per i dati pertinenti alle singole viste; un ultimo mostra la lista contatti ed un insieme di funzionalità minimali per la chat.
Al di sopra dell'interfaccia vera e propria si trova una barra di ricerca, che consente di formulare query contestuali alla vista selezionata.
La qualità che permette a Gmail una tale essenzialità grafica e un'estrema facilità d'utilizzo è una studiata tipizzazione dei dati rappresentati.
Il concetto di mail è stato sostituito da quello più potente di conversazione, quello di destinatario con quello di contatto. In questo modo è possibile un'ottima integrazione e omogeneità tra viste differenti: si può agire nell'ambiente chat come si farebbe in quello mail, passare da una conversazione in tempo reale ad una asincrona, e viceversa.

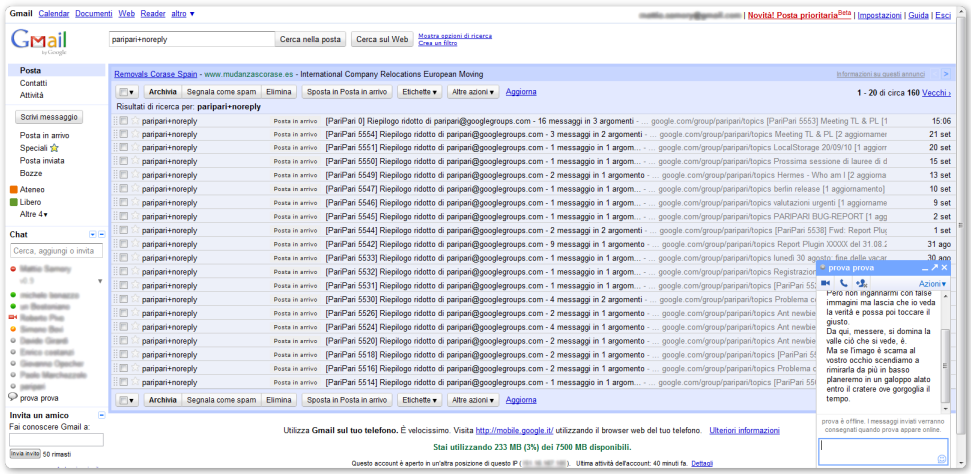

Figura 2.5: *Interfaccia grafica di Gmail*

2.4 Confronto

PariPari pone la particolare sfida di integrare tutte le funzionalità offerte dalle applicazioni brevemente analizzate, e molte altre ancora, in maniera dinamica.

Si è visto come BitTorrent in confronto ad eMule riesca a sintetizzare un'interfaccia più leggera ed essenziale e si focalizzi su ciò che è effettivamente più rilevante per l'utente nella maggior parte dei casi: mantiene in primo piano i file in trasferimento, i filtri e le operazioni per operare su di essi, mentre dettagli e statistiche assumono una posizione secondaria.
In PariPari però non è definibile una funzione principale, utilizzabile come centro attorno al quale sviluppare una GUI, in quanto ogni plug-in ne apporta di nuove e differenti, in quantità non definibili a priori.
Non solo: anche limitandosi alla rappresentazione per ogni plugin di una sola funzionalità principale, il formato necessario per la visualizzazione non può essere ridotto indefinitamente. Basti pensare alla tabella dei risultati di eMule, a quella dei file in trasferimento di BitTorrent, alla lista contatti di Skype, a quella delle conversazioni di Gmail.

Skype ottimizza gli spazi adattandoli in risposta alle azioni compiute dall'utente e prediligendo l'impiego di simboli sul testo.
Il primo di questi metodi è attuabile solo fintantoché il numero delle azioni possibili è contenuto: il rischio che si corre altrimenti è quello di creare un ambiente eccessivamente instabile, visivamente affaticante e difficilmente gestibile.

Lo stesso ragionamento vale per il secondo: se il set di simboli presentati è troppo esteso, il discernimento tra questi diventa meno immediato, e le corrispondenze con i significati vengono interiorizzate con maggiore difficoltà.
Oltre a ciò, compare anche il rischio di incorrere in interpretazioni incongruenti di uno stesso simbolo.

Gmail consente di riutilizzare i componenti dell'interfaccia in contesti differenti, grazie ad un'astrazione dei concetti rappresentati, ed un'unificazione delle modalità d'interazione con essi.
Non sempre però questo è possibile. Per le caratteristiche del protocollo BitTorrent, ad esempio, non è prevista la possibilità di ricercare direttamente dall'applicazione client. Similmente, il "contatto" in eMule è qualificato per mezzo di attributi peculiari alla rete in cui si trova, e risulta molto differente quindi dal corrispettivo in Gmail o Skype.

Le interfacce portate ad esempio, così come quelle della maggior parte dei prodotti software indirizzati al mercato consumer, sono ottimizzate per rappresentare realtà particolari e circoscrivibili. La multifunzionalità di PariPari rende dunque indispensabile un approccio inedito per la progettazione di una sua interfaccia.

Capitolo 3

Design ed ergonomia

PariPari è una piattaforma ricca, flessibile, impiegabile in ambiti assai diversi tra loro. Molte sono le funzioni da rappresentare, molte quelle che ancora non sono state definite, grazie alla sua particolare struttura a plug-in. La difficoltà primaria che si è incontrata nell'ideazione di una interfaccia è stata la ricerca di una rappresentazione semplice ed intuitiva, ma al contempo sufficientemente versatile da consentire la futura estensione della realtà riprodotta. In questo capitolo vengono analizzate le considerazioni e le scelte fatte per quanto riguarda il design e le motivazioni che ne stanno alla base.

3.1 Difficoltà nel design

Alan Cooper imputa il fallimento delle interfacce grafiche di molti prodotti digitali a tre cause principali: la scarsa comprensione delle esigenze e delle motivazioni degli utenti, il conflitto degli interessi di sviluppatori e designer e la mancanza di un processo produttivo.[1]

Incomprensione dell'utente

Gran parte del software giunge alla fase di rilascio senza che venga effettuata ricerca su quelli che ne saranno gli utilizzatori. Al più, vengono effettuati studi di settore che portano alla modellazione dei potenziali clienti in segmenti di mercato ricavati sulla base di parametri economici e socio-demografici, poco indicativi delle loro effettive necessità e motivazioni. Lo studio dell'ergonomia viene spesso incluso solamente negli ultimi stadi della catena di produzione, come riparo all'intrinseca complessità dell'applicazione. Per poter effettivamente an-

[1] Cfr.: [5]

dare incontro alle necessità degli utenti, questa fase di progettazione dovrebbe precedere la programmazione.

Conflitto di interessi

Si possono individuare due tendenze che si contrappongono durante lo sviluppo di un'interfaccia: da una parte si ha la necessità di agevolare gli utenti nel perseguire i propri obiettivi, dall'altra si devono rispettare precise priorità costruttive. Il considerevole investimento di tempo richiesto dalla progettazione può scontrarsi con il rispetto di scadenze dettate dall'evoluzione repentina del mondo del software. In molti casi, essendo affidati ai programmatori i compiti di designer e di responsabile del marketing, questi possono effettuare scelte in maniera non disinteressata, o condizionata dalle particolari attitudini e competenze. Un prodotto è tanto più desiderabile quanto più elevata è la sua usabilità, ed al contempo quanto più ricche sono le caratteristiche che offre in confronto a quelle dei concorrenti.

Mancanza di processi

In ultimo, data la comparsa relativamente recente del settore, non si sono ancora affermati processi produttivi per l'ideazione di interfacce. Per quanto si tratti di un argomento dibattuto, mancano a tutt'oggi modelli rigorosi e di comprovata validità in grado di garantire efficienza nello sviluppo, realizzabilità del progetto e soddisfazione dell'utente. Allo stesso modo, sono in via di definizione nuove figure di riferimento e i confini tra *visual*, *communication* ed *interaction design*.

La costituzione e l'organizzazione interna del progetto PariPari poneva in varia misura tali problematiche. E' stato quindi necessario individuare criteri per l'analisi e strumenti procedurali per la creazione di una GUI.

3.2 Conseguenze

Le conseguenze di un cattivo studio dell'interazione con l'utente sono molteplici, e sono spesso cofattori, o fattori primari dell'insuccesso commerciale di prodotti software.
Interfacce mal progettate presentano caratteristiche riconoscibili:

- *Perdita in efficacia:* risulta difficile per l'utente raggiungere i propri obiettivi in maniera accurata e completa. Questo avviene ad esempio se le funzioni messe a disposizione dall'interfaccia non corrispondono a quelle richieste, o non sono sufficientemente potenti da portare ai risultati desiderati.

- *Diminuzione dell'efficienza:* vengono impiegate troppe risorse per il raggiungimento di uno scopo. Se lo sforzo richiesto per compiere una determinata azione è sproporzionato alle aspettative, l'applicazione può essere vista più come un intralcio che come un supporto.

- *Insoddisfazione:* l'utente perde motivazione nell'utilizzo del programma. L'esperienza che deriva dall'interazione è frustrante, svalutante, ed impedisce di lavorare in condizioni accettabili.

- *Difficoltà di apprendimento:* molte interfacce hanno curve di apprendimento troppo lente, non consentono all'utente di raggiungere buone prestazioni in tempi ragionevoli. Sebbene in alcuni casi sia necessaria una conoscenza approfondita dell'applicazione, e di conseguenza sia indispensabile impiegare un periodo abbastanza lungo per imparare l'uso delle funzionalità avanzate, la maggior parte delle volte è sufficiente la padronanza degli elementi di base: questi dovrebbero essere compresi rapidamente, per permettere all'utente di essere operativo nel più breve tempo possibile.

- *Difficoltà di memorizzazione:* dopo un periodo di lungo inutilizzo, l'utente per poter interagire nuovamente con l'interfaccia è costretto a ricominciare da zero.

- *Propensione all'errore:* una poco accurata disposizione degli elementi sui quali agisce l'utente può indurlo a commettere errori: se, ad esempio, ad un componente che ha conseguenze rilevanti sul funzionamento del programma viene assegnata una posizione centrale o prossima ad altri componenti di uso frequente, è probabile che venga involontariamente azionato.

3.3 Programmazione e visualizzazione

Non solo la fase di studio di ergonomia ed usabilità dovrebbe precedere quella di programmazione; adottare per il design dell'interfaccia gli stessi schemi utilizzati per la definizione della struttura interna dell'applicazione porta, nella maggior parte dei casi, a cadere in uno o più errori tra quelli elencati.
Adattibilità, predisposizione al *problem solving* ed al pensiero matematico sono capacità che fanno parte del ventaglio di strumenti a disposizione dello sviluppatore, ma che non dovrebbero essere richieste all'utente.
E' molto probabile che l'atteggiamento con il quale il creatore del programma si pone nei confronti del suo prodotto risenta delle sue competenze, anche quando cerchi di

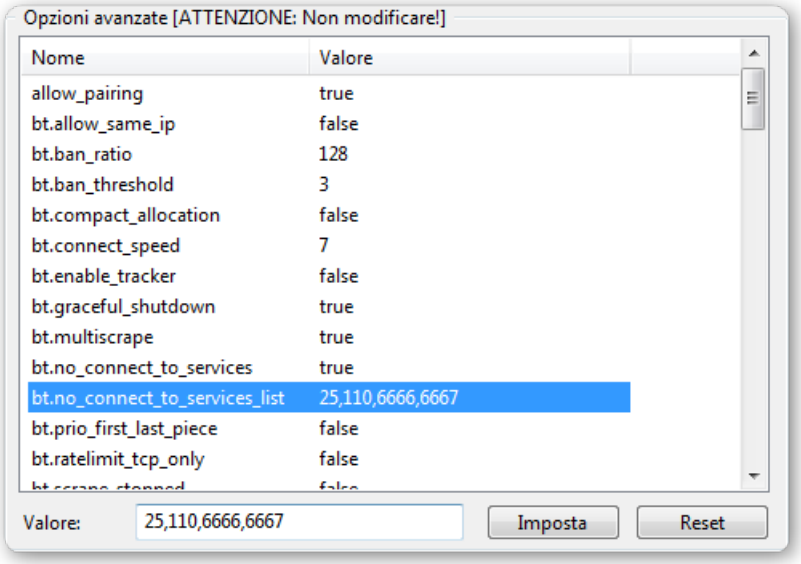

Figura 3.1: *Il pannello delle impostazioni avanzate di µTorrent presenta molte delle caratteristiche negative riportate nel paragrafo 3.2, tantoché scoraggia esplicitamente l'interazione con l'utente*

immedesimarsi nell'utente inesperto, dal momento che ne ha interiorizzato la logica e le peculiarità.
A chi conosce i dettagli di come effettivamente funzionano l'applicazione ed il supporto in cui questa viene eseguita, può venire naturale tradurre questi in un design che ne rifletta il comportamento, fornire quindi semplicemente una mappa di funzioni, transazioni, opzioni.
Nel loro lavoro, gli sviluppatori spesso dimostrano di pensare al sofware come ad una lista di specifiche e caratteristiche, poiché la codifica avviene utilizzando la funzione come modulo.
Sono molti gli esempi in cui si vedono interfacce sovraffollate di bottoni, senza che i rapporti tra questi rispecchino la relativa categoria d'appartenenza o la frequenza d'utilizzo, o che costringono l'utilizzatore a concentrarsi su numerose pagine di finestre di dialogo gremite di termini a lui oscuri.
Il motivo per cui tale approccio non è adatto all'ideazione di *UI* è che non tiene conto del modello di pensiero dell'utente, non ne rispetta le sintassi e le modalità d'apprendimento.

3.4 Modello

Per poter operare su un programma, l'utente non deve necessariamente comprendere il suo funzionamento interno, ossia come il codice compia effettivamente le operazioni che portano dalla richiesta al risultato.
La mente umana tende a creare modelli della realtà più semplici, ad astrarre dal dettaglio per formare schemi di massima, complessi quanto basta per riassumere le caratteristiche di interesse. Tali immagini possono addirittura essere analogiche più che sintetiche. Donald Norman definisce queste interpretazioni della realtà *"mental model"*, modelli mentali.[7]
Ad acuire la difficoltà di comprensione dei meccanismi di funzionamento dei prodotti software, in paragone a quelli industriali, è la mancanza di un supporto visivo intrinseco, una meccanica tangibile e sperimentabile fisicamente.

Da parte loro, i prodotti software offrono una flessibilità, in passato sconosciuta, di separare la realizzazione dalla rappresentazione.
Non bisogna pensare che la scarsa veridicità di tali modelli li renda descrizioni meno efficaci in pratica: l'utilizzo di *"black box"*, destinate a nascondere particolari non essenziali, permette di ottimizzare gli sforzi cognitivi e di concentrarsi solo sugli obiettivi.

Sebbene presenti una situazione patologica, Italo Svevo riesce a trasmettere perfettamente, amplificato dal pensiero ipocondriaco di Zeno Cosini, il disagio provocato dal dover tenere conto di informazioni irrilevanti:

> *"Aveva studiato l'anatomia della gamba, e del piede. Mi raccontò ridendo che quando si cammina con passo rapido, il tempo in cui si svolge un passo non supera il mezzo secondo e che in quel mezzo secondo si muovevano nientemeno che cinquantaquattro muscoli. Trasecolai e subito corsi col pensiero alle mie gambe a cercarvi la macchina mostruosa. Io credo di avercela trovata. Naturalmente non riscontrai i cinquantaquattro ordigni, ma una complicazione enorme che perdette il suo ordine dacché io vi ficcai la mia attenzione.*
> *Uscii da quel caffè zoppicando e per alcuni giorni zoppicai sempre. Il camminare era per me divenuto un lavoro pesante, e anche lievemente doloroso.,,* [2]

Non è necessario quindi che l'utente cambi il proprio modo di pensare per uniformarsi a quello del programmatore.

[2]Estratto da *"La coscienza di Zeno"* di Italo Svevo, edito per la prima volta nel 1923

Un'applicazione che offre un'interfaccia vicina al modello mentale degli utenti, con la quale questi possono interagire in maniera naturale, risulta facile da usare, offre un'esperienza piacevole, permette un apprendimento rapido ed una memorizzazione duratura dei suoi meccanismi di funzionamento.

3.5 Stile

Lo stile che deve contraddistinguere visivamente PariPari è minimale, pulito, non invasivo. Ciò è indispensabile per presentare in maniera chiara la moltitudine delle funzionalità offerte. Viene di seguito riportata un'analisi delle scelte fatte riguardo agli elementi figurativi di base per ottenere l'effetto desiderato.

Linea La linea utilizzata è sottile, continua, morbida, poco marcata. In questo modo la struttura della composizione risulta più leggera, viene posta l'enfasi sul contenuto.

Spazio Si suppone che durante l'uso di PariPari l'utente vi concentri unicamente e per lunghi periodi la sua attenzione. L'interfaccia deve dunque coprire l'intera area dello schermo. Data la complessità dell'applicazione, tale spazio non è sufficiente per poterla rappresentare interamente; per questo si è deciso di dividere i contenuti in pannelli sovrapposti, più propriamente *tab*, in modo che le informazioni necessarie alle attività relative ad una particolare categoria semantica siano per quanto possibile visibili contemporaneamente. I criteri di tali divisioni saranno discussi in seguito, nella sezione 3.8.2.

Peso visivo Si è scelta una composizione asimmetrica. Nonostante i plug-in siano per il programma paritetici, non era opportuno riportare visivamente questo fatto. Alcune funzioni per essere comprensibili ed utilizzabili devono poter mostrare informazioni in quantità maggiori, o in formati più ingombranti. Conseguentemente, aree più estese sono state assegnate a compiti che ne richiedono di più. Questa assunzione ha avuto conseguenze radicali nell'intero design.

Colore L'interfaccia è sostanzialmente monocromatica, per non appesantirne la percezione e distrarre l'utente. Per lo stesso motivo si è scelto un colore, il verde, né caldo né freddo.

Forma Il modulo principale è un rettangolo arrotondato, che conferisce staticità ma non pesantezza, un ibrido di geometrico ed organico. Inoltre, sfrutta al meglio il formato dello schermo.

Texture In linea con lo stile discreto adottato, l'effetto è bidimensionale, disegnato, piatto, liscio, omogeneo. La tridimensionalità, così come la lucidità, è minimizzata.

Intensità Il colore è tenue, desaturato, non aggressivo: in questo modo anche in caso di un'esposizione prolungata la vista non ne risulta affaticata.

3.6 Principi

Così come per la forma, sono emersi dei principi generali per la definizione delle modalità d'interazione.

Intuitività

L'interfaccia deve cercare di assecondare il senso comune, ove ciò non limiti le sue stesse potenzialità. Questo non implica necessariamente adattarsi agli schemi più popolari, ma individuare gli approcci che meglio rispecchiano il modello mentale degli utenti.

La stessa interfaccia deve inoltre presentare una forte coerenza interna, si devono poter effettuare operazioni simili con modalità simili.

Le funzioni più rilevanti o frequenti devono risultare facilmente raggiungibili, posizionate adeguatamente e sufficientemente evidenziate. In questo modo migliorano tempi di apprendimento, efficienza d'utilizzo, soddisfazione dell'utente.

Minimalità

Si desidera fare in modo che l'utente sia in grado di adoperare tutte le funzionalità di PariPari con il minimo set di elementi.

Potenziare i singoli componenti comporta il rischio di sovraccaricarli di significati, facendogli quindi perdere l'*intuitività* originaria. Pertanto bisogna porre particolare attenzione, ancora una volta, alla semantica che intercorre tra le funzionalità fornite, secondo la logica proposta.

Graficità

Si vogliono esprimere al meglio le potenzialità dell'interfaccia grafica. La mente umana riconosce più rapidamente i simboli rispetto al testo, l'azione è più efficace della descrizione della stessa.

Ove possibile si dovrebbe permettere all'utente di agire direttamente sul componente grafico, ad esempio attraverso l'utilizzo di meccanismi di *drag-and-drop*.

Immediatezza

Per ottenere una buona efficienza bisogna mettere in grado l'utente di arrivare

al proprio obiettivo con il minimo numero di operazioni. Per ottenere questo da una parte è essenziale una rapida risposta dell'interfaccia agli input, dall'altra è necessario che ogni operazione sia composta dal numero minimo di fasi.
Ad esempio, il ricorso alle finestre di dialogo dovrebbe essere il più possibile evitato, e dovrebbe essere limitato il numero di schermate che le compongono.

Discrezione

Il flusso d'azione dell'utente non dovrebbe essere interrotto se non quando ciò risulta inevitabile.
Notifiche non critiche dovrebbero non essere omesse, ma riportate in zone dello schermo non centrali, possibilmente in prossimità al componente che le ha generate.
Pop-up, finestre modali, segnali acustici e altre potenziali fonti di distrazione dovrebbero essere usati con parsimonia.

3.7 Target

Le funzionalità offerte da PariPari trovano applicazione tanto in contesti ricreativi quanto professionali. Bisogna quindi conciliare le esigenze di utenti con livelli di esperienza molto differenti tra loro.

Gli utenti possono essere ricondotti in linea di massima a tre categorie, in base al grado di conoscenza del programma: novizi, intermedi ed esperti.

I novizi non sanno quali azioni possano compiere attraverso l'applicazione; devono per la prima volta interiorizzare i simboli e le procedure per interagire con essa. Questi utenti necessitano di particolari meccanismi in grado di introdurli alla logica del programma, corredati di spiegazioni sufficientemente chiare e estese.
Tipicamente, i novizi non rimangono nel loro stato a lungo, ed avanzano verso il livello intermedio, per non tornare più indietro. Gli strumenti forniti per i principianti risultano quindi utilizzati per un breve periodo, successivamente al quale diventano inutilmente prolissi.

Gli utenti esperti hanno una conoscenza dettagliata delle potenzialità dell'applicazione. Desiderano di conseguenza poter personalizzare l'area di lavoro, sfruttare funzionalità avanzate, automatizzare le operazioni più frequenti. Il controllo di basso livello da loro richiesto però è troppo specifico per gli utenti meno pratici.

La maggioranza degli utenti è costituita dagli intermedi, che hanno familiarità con le caratteristiche del programma e riescono ad eseguire i compiti più comuni con una certa dimestichezza.
Saltuariamente, se messi nella condizione di farlo, possono diventare esperti, a seguito ad esempio di un uso prolungato dell'applicazione; viceversa, in caso di un temporaneo distacco, utenti esperti possono perdere confidenza con l'applicazione e tornare ad un livello intermedio.

E' quest'ultima categoria che si intende inquadrare come target, seppur venendo incontro alle necessità delle altre. Si favorirà l'accesso alle funzionalità più comuni, che verranno presentate in maniera il più possibile concisa ed efficace. Per permettere agli utenti inesperti di progredire, per i primi accessi si forniranno tutorial che permettano di scoprire come interagire con l'applicazione, attraverso esempi pratici invece che "wizard". Gli utenti più smaliziati avranno estese possibilità di configurazione, per avere a portata di mano le opzioni che usano più di frequente.

3.8 Realizzazione

3.8.1 Svantaggi di un'interfaccia modulare

La decisione forse più radicale nel design dell'interfaccia per PariPari è stata il voler prescindere dalla sua natura modulare. La sua architettura a plug-in comporta una significativa incertezza sulla quantità e qualità delle informazioni da rappresentare. L'idea di più immediata realizzazione sarebbe stata l'adozione di una struttura altrettanto componibile, che potesse adattarsi a ogni variazione del set di componenti del client. Questo potrebbe essere realizzato attraverso un telaio statico su cui montare e smontare sotto-interfacce dedicate al singolo plug-in. Tuttavia, percorrere questa via porta a diverse implicazioni negative: l'eccessiva mutabilità, l'indifferenziazione tra i plug-in, e la replicazione di funzionalità.

- L'incostanza aliena l'utente. L'introduzione di moduli obbliga l'utente ad adattarsi a composizioni sempre nuove. Le poche costanti nell'interfaccia, dedicate alla gestione dei plug-in, riguardano funzionalità ad uso infrequente. Questa incertezza è emotivamente frustrante e cognitivamente dispersiva.

- Ogni plug-in è trattato allo stesso modo. Ad ognuno viene attribuito lo stesso peso e lo stesso spazio, indipendentemente dalla sua rilevanza relativa. Non viene

tenuto conto del fatto che un plug-in possa richiedere un'interazione con l'utente più o meno breve; ad esempio, tipicamente si dedica ad un programma di *file sharing* il tempo strettamente necessario alla ricerca, dopodiché questo agisce in *background*, mentre un'applicazione di *instant messaging* assorbe l'utente per l'intera durata di una conversazione.

- Viene introdotta ridondanza. I punti di forza di PariPari sono la sua multifunzionalità e la sua capacità d'integrazione con le reti già esistenti. I suoi benefici sono pienamente ottenibili solo se l'utente è in grado di fruire di *tutti* o di buona parte di questi servizi: si presuppone che se l'utente desidera scaricare un file, sia poco interessato a *quale* rete venga utilizzata in particolare, e che rimanga per lui più importante la rapidità con cui riesce ad ottenere il file.

Tale approccio porta dunque ad un uso poco oculato dello spazio visivo e simbolico a disposizione. Si è cercato quindi di trovare una maniera per accomunare le rappresentazioni dei plug-in. Si voleva conferire all'interfaccia una maggiore stabilità, che fornisse un ambiente al contempo facilmente comprensibile, potente e completo.

3.8.2 Soggetti, aree funzionali, azioni

Si è cercato in particolar modo di presentare una divisione delle funzionalità che rispecchiasse il modo di pensare dell'utente, unificasse modalità operative ed oggetti d'azione per quelle funzionalità intuitivamente riconducibili ad una categoria comune. Il modello che ne è derivato è composto di tre entità fondamentali: i *soggetti*, le *aree funzionali* e le *azioni*. I *soggetti* sono gli individui con i quali è possibile relazionarsi attraverso le azioni, le *aree funzionali* stabiliscono la semantica con la quale si partecipa alla condivisione, le *azioni* le specifiche modalità di interazione.

A ciascun elemento è stata attribuita una porzione di schermo. Le azioni sono nettamente più ricche di contenuti, essendo il luogo dove risiedono le funzionalità dell'applicazione. Sono state quindi posizionate al centro e a destra, per un'estensione che copre la maggior parte della superficie disponibile. Le aree funzionali, che pongono la distinzione tra le categorie di azioni, possiedono lo spazio immediatamente superiore a queste. I soggetti occupano la colonna a sinistra delle azioni.

Stabilire un criterio di discernimento delle aree funzionali, che fosse sufficientemente selettivo per garantire una buona usabilità, ma al contempo potesse consentire l'inclusione futura di nuovi plug-in, è stato un altro aspetto critico nella progettazione.

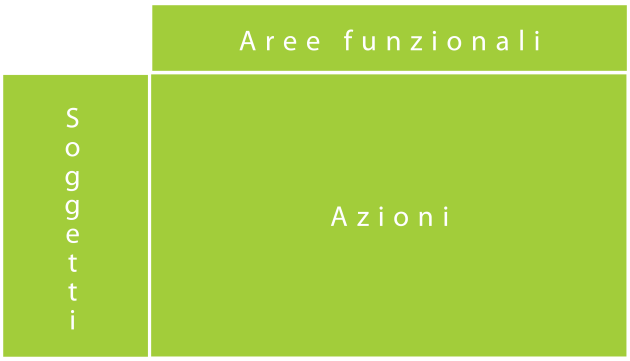

Figura 3.2: *La divisione dello schermo avverrà secondo la ripartizione degli elementi in soggetti, aree funzionali ed azioni*

I plug-in che richiedono controllo da parte dell'utente attualmente in via di sviluppo sono stati lo spunto per una divisione in due classi: *Search&Share* e *Connect*.
In *Search&Share* troveranno sede i fornitori di servizi di file sharing in senso lato, come `Mulo` e `Torrent`, ma anche `DistributedStorage`.
Connect comprende le funzionalità mirate alla comunicazione, come quelle messe a disposizione da `Im`, `Voip`, `IRC`.
Per la rappresentazione grafica di tali categorie, si è optato per delle etichette, a selezione esclusiva: aree funzionali ed azioni formano così un *tabbed panel*, un componente che gli utenti sono abituati ad usare, in particolar modo nell'ambiente web.
Sebbene questi gruppi siano sufficientemente ampi da consentire l'inclusione di molti dei servizi desiderabili, a seconda delle esigenze sarà possibile integrare nuove funzionalità semplicemente aggiungendo nuove categorie.

La separazione non è perfetta: esistono casi nei quali si vorrebbe poter usufruire dei servizi appartenenti ad una sfera mentre si sta compiendo un'azione in un'altra; ad esempio, la condivisione di file è una funzionalità solitamente accessibile anche nel corso di una chat. Crediamo però che questo non invalidi la divisione proposta, in quanto la duplicazione di funzionalità secondarie in un'area non strettamente pertinente non vada contro il senso comune degli utenti.

3.8.3 Coerenza funzionale

Grazie alla categorizzazione adottata, è stato possibile dare una definizione comune, più generale, di funzionalità ed elementi: in questo modo non verranno percepiti molti dei dettagli realizzativi, permettendo un approccio all'interfaccia più naturale ed immediato.

Si è deciso di rendere l'impiego di protocolli differenti nella comunicazione trasparente all'utente. E' stato quindi proposto il concetto di "contatto", in modo da comprendere solo gli attributi generali che ne permettono l'identificazione, la risoluzione dello stato, il raggiungimento attraverso le modalità di comunicazione disponibili.

La colonna dei *soggetti* è formata dunque dalla lista contatti, appartenenti a tutti i servizi che ne prevedono l'esistenza. Per organizzare i contatti a seconda delle relazioni stabilite è prevista una gestione per gruppi: per comprendere anche contatti all'utente sconosciuti, una scomposizione possibile è quella per "distanza affettiva": *io-conoscenti-mondo*, dove *conoscenti* e *mondo* possono essere a loro volta formati da sotto-gruppi non necessariamente disgiunti.
Per comunicare con un contatto attraverso il mezzo preferito, sarà sufficiente portarlo, con un'operazione di *drag-and-drop*, sull'etichetta del tab Connect. Verrà dunque visualizzato il pannello relativo, dove una nuova sessione del servizio richiesto sarà stata attivata. Alternativamente, selezionando il contatto apparirà un menù a tendina con le opzioni disponibili.

Similmente a quanto stabilito per i contatti, sono state unificate le definizioni di file per il trasferimento in una più generale, slegata dalla rete dalla quale possa essere reperito. Vengono presentate solamente le caratteristiche comuni alle differenti realizzazioni, in quanto forniscono tutte le informazioni effettivamente rilevanti per l'utente. I file in trasferimento ed i risultati delle ricerche di tutti i plug-in riferibili al pannello Search&Share saranno visualizzati in una singola tabella, senza diversificazioni se non per stato (inattivo, in download, in upload) e luogo di memorizzazione (locale o remoto).
In maniera simile a come avviene per la comunicazione, per inviare un file ad un contatto sarà sufficiente trascinare il file sullo stesso.

3.8.4 Uniformità delle modalità d'interazione

Oltre ad una coerenza relativa alle funzionalità, ossia l'adozione di una semantica vicina al modello mentale dell'utente, si è ricercata una parificazione delle modalità d'interazione a livello di tab.

Uno dei risultati di tale studio è la cosiddetta *"bolla di ricerca"*.
E' questo il principale strumento messo a disposizione in ogni tab, situato nella parte superiore di ogni pannello. E' composta da un'area di testo nella quale gli utenti potranno formulare query, e da due famiglie di filtri.

A sinistra, rappresentati da icone, vi sono i filtri statici, utilizzati per limitare il campo d'azione delle ricerche. Nel pannello Search&Share, ad esempio, uno di questi permette di individuare solamente i file presenti sulla macchina sulla quale è in esecuzione il client, mentre un altro permette di selezionare i soli file in trasferimento.
A destra, invece, si trovano i filtri dinamici, costituiti da etichette. Questi vengono creati e rimossi dinamicamente in relazione alle azioni dell'utente e sono adoperati per controllare cosa venga visualizzato all'interno del pannello.
Nel tab Connect corrispondono ai servizi (chat, mail, video-conferenze etc.) attivi in uno specifico istante; poiché il pannello consente di visualizzare fino ad un massimo di quattro componenti contemporaneamente, al più quattro etichette saranno selezionate, mentre i quadranti liberi verranno occupati dai rispettivi *widget*.
Nel tab Search&Share, tali filtri fungono da *entry* della cronologia di ricerca, memorizzando le parole chiave delle query passate. Per riportare una lista di risultati precedente nella tabella dei file basterà selezionarne l'etichetta.

Molti sono i dettagli ancora in via di definizione per quanto riguarda il design dell'interfaccia grafica; il lavoro necessario all'integrazione di nuove funzionalità probabilmente non avrà mai termine, così come la ricerca di ergonomia ed usabilità migliori. Le linee guida per tali sviluppi, presentate in questo capitolo, sono però ben definite, e rispecchiano ciò che si pensa debbano essere l'aspetto ed il comportamento di PariPari.

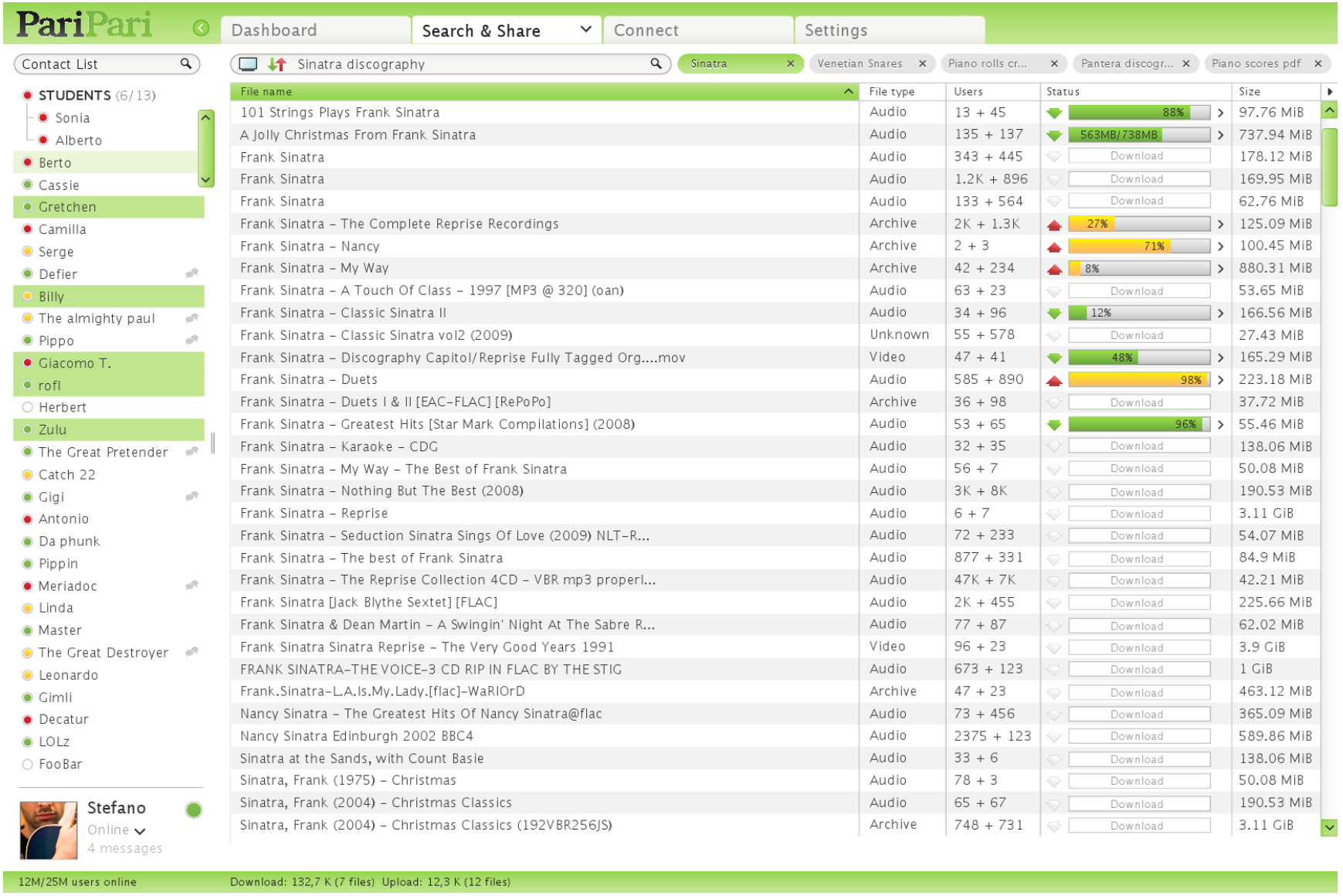

Figura 3.3: *Anteprima del pannello "Search&Share" realizzata da Stefano Calgaro*

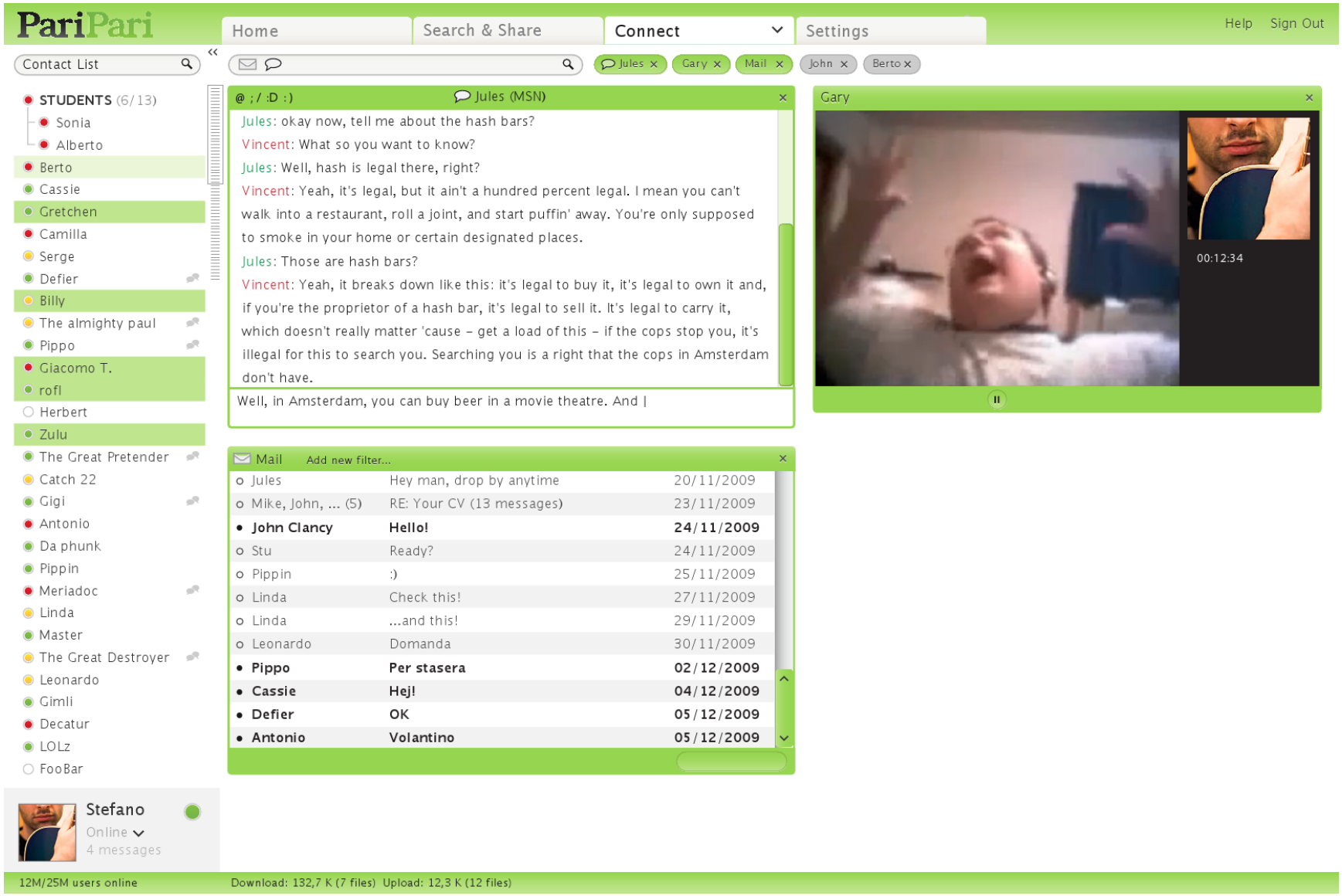

Figura 3.4: *Anteprima del pannello "Connect" realizzata da Stefano Calgaro*

Capitolo 4

Realizzazione ed interfacciamento

Come esposto nel precedente capitolo, la visualizzazione di PariPari si discosta fortemente dalla struttura interna dell'applicazione. Per poter realizzare l'interfaccia utente come plug-in è necessario introdurre un livello intermedio tra queste due realtà, che ne permetta l'integrazione.

4.1 La comunicazione in PariPari

Riassumendo i concetti chiave riportati nella sezione 2.2.2, PariPari presenta un'architettura modulare, che permette di aggiungere facilmente alla piattaforma nuovi servizi. Il nucleo operativo, indispensabile per il funzionamento del client, è costituito dal `Core`, che rende possibile la coesistenza dei componenti aggiuntivi, detti `Plugin`. La tecnologia Java Web Start consente al `Core` di caricare e tenere sempre aggiornati i componenti di PariPari a *runtime*, sia in locale che da remoto.[1]

Per potersi integrare con il resto del client, i `Plugin` devono attenersi a meccanismi di estensione e comunicazione ben definiti, contenuti nel package `paripari.plugin`. L'interazione tra `Plugin` e l'accesso di questi a risorse critiche sono soggetti a determinate limitazioni: poiché chiunque può creare un nuovo componente per PariPari, è facile immaginare come senza uno stretto controllo sulla sicurezza, possano intervenire rischi di manomissioni ad opera di malintenzionati.

In primo luogo, i `Plugin` non hanno modo di comunicare direttamente tra di loro né con il mondo esterno; non solo, questi *non sono a conoscenza dell'esistenza di altri*

[1] Cfr.: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136112.html

Plugin. Può essere supposta solo la presenza dei gestori di risorse di base della cerchia interna:

`DHT`, che permette la partecipazione alla rete di PariPari,

`Credits`, che amministra il sistema di crediti interno a PariPari,

`Connectivity`, che fornisce gli strumenti di base per la connettività,

`LocalStorage`, che gestisce l'accesso al file system.

Non potendo rivolgersi direttamente agli altri `Plugin`, si è limitati a richieste di risorse, tra quelle definite dalle interfacce presenti nel package `paripari.API`. Al momento del caricamento di nuovi moduli, per ognuno di questi il `Core` si preoccupa di tenere nota di quali realizzazioni delle `API` questi forniscano, e di controllare che le risorse indispensabili per il loro corretto funzionamento (*dependencies*, "dipendenze") siano messe a disposizione da qualche altro `Plugin`, attivo o attivabile.
L'unico interlocutore al quale inoltrare le richieste di risorse è dunque il `Core`: sarà questo a recuperare la realizzazione che riterrà più adatta tra quelle offerte dai `Plugin` caricati. A stabilire i criteri per scegliere quale realizzazione favorire nel caso ne sia presente più d'una, e se un `Plugin` ha diritto a tale risorsa, è il sistema dei crediti, che quindi inserisce ulteriori limitazioni nelle transazioni interne a PariPari.

Per fare un esempio pratico del meccanismo di comunicazione, supponiamo che il `Plugin TextEditor` sia un elaboratore di testi; per poter funzionare deve poter leggere e scrivere liberamente da file, ma non ha accesso diretto al file system. Deve quindi chiedere al `Core` di procurargli un oggetto che realizzi `FileAPI`. Il `Core` valuta se è possibile soddisfare la richiesta: controlla innanzitutto che esista almeno un `Plugin` che ne offra una realizzazione. Scopre che `Plugin1` e `Plugin2` mettono a disposizione allo scopo rispettivamente `StorageFile1` e `StorageFile2`: tra le due realizzazioni, quella preferibile, secondo i criteri stabiliti dal sistema, è la prima. Poiché i crediti a disposizione di `TextEditor` sono sufficienti a permettergli di ottenere la risorsa, il `Core` esegue la transazione, acquisendo uno `StorageFile1` da `Plugin1`, risvegliando il thread richiedente, ed aggiornando il bilancio crediti dei due `Plugin`.

`TextEditor` *possiede ora un oggetto che realizzi* `FileAPI`*, ma non ne conosce i dettagli, né il fornitore.*

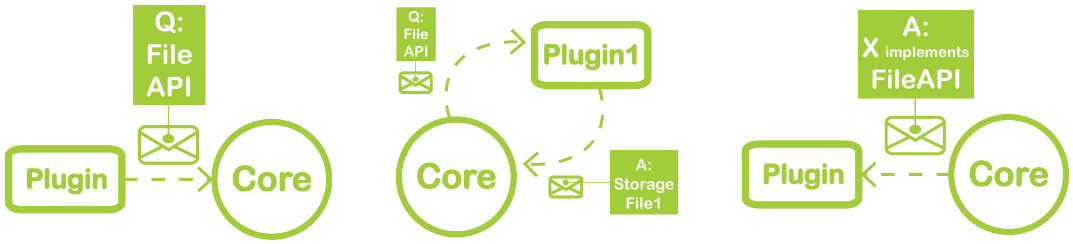

Figura 4.1: *Illustrazione esemplificativa della comunicazione all'interno di PariPari. Un plug-in domanda una risorsa di tipo "FileAPI" al Core, il quale ne ricerca una realizzazione, che restituisce al richiedente*

4.2 Interfacciamento con la GUI

La modalità d'interazione per richieste di `API` al `Core` conferisce a PariPari un'enorme flessibilità. I `Plugin` possono infatti essere sviluppati in totale indipendenza, senza doversi preoccupare di modifiche nei gestori delle risorse, grazie al livello d'astrazione interposto tra cliente e fornitore. Il `Core` assume il ruolo di intermediario autorevole al quale i `Plugin` si rivolgono con un approccio di tipo "a scatola nera".

Sebbene questo comportamento sia desiderabile nelle normali comunicazioni interne a PariPari, mal si presta all'integrazione di una interfaccia grafica. Il solo fatto di *non sapere quali plug-in siano presenti in un determinato momento*, essendo anch'essa un plug-in, è problematico. Generalmente, infatti, per quanto dinamicamente modificabili, le *UI* (*User Interface*, interfacce utente) sono la rappresentazione della realtà strutturalmente statica di un programma. Per natura della piattaforma, invece, questa non può essere assunta come invariante: non è il solo stato dell'applicazione a cambiare durante l'esecuzione, ma l'applicazione stessa.

4.2.1 Un primo approccio

A differenza dei membri della cerchia interna, con i quali ugualmente si suppone debbano interagire tutti i `Plugin`, si può affermare che la GUI necessita di conoscere esattamente quali tra questi sono disponibili. Per rispecchiare nel modo più fedele possibile la struttura modulare di PariPari, si era pensato in prima battuta di offrire ai `Plugin` degli strumenti per la definizione di una propria interfaccia; gestire la composizione di queste ultime sarebbe stato poi compito della GUI.

Come *UIDL* (*User Interface Description Language*, linguaggio per la descrizione di interfacce utente) si era pensato di adottare l'*XML* (*eXtensible Markup Language*), per la sua intrinseca flessibilità. Attraverso la stesura di un documento contenente *tag* a livello di componente grafico, ogni `Plugin` avrebbe avuto la possibilità di strutturare la propria porzione di interfaccia nei minimi particolari.[2]
Grazie alla genericità delle definizioni contenute nel documento XML, e data la relativa omogeneità dei componenti grafici offerti dalla maggior parte dei linguaggi di programmazione per interfacce grafiche, si sarebbe fornito supporto per visualizzazioni multiple. Da uno stesso modello si sarebbero potute creare interfacce basate su temi differenti, in linguaggi differenti, ottimizzate per i più diversi dispositivi. Questa versatilità sarebbe stata potenziata attraverso la possibilità di rendere remota la GUI, ossia in grado di controllare un'istanza del client in esecuzione su un'altra macchina.

Per un'analisi approfondita della progettazione secondo questa visione, si rimanda alla tesi di Simone Vidotto [4]. In sostanza, si volevano creare dei plug-in visuali, fornendo una mappa 1:1 della realtà sottostante.

4.2.2 Problematiche legate alla realizzazione

Le possibili conseguenze dal punto di vista del design implicate da una scelta simile sono state trattate nella sezione 3.8.1; si presenteranno qui le problematiche emerse riguardo alla realizzazione.

Al momento di scrivere il codice per una GUI secondo la struttura proposta, si sono dovute fronteggiare diverse difficoltà. In primo luogo, le dimensioni del documento XML sono cresciute al di là delle previsioni per la descrizione di composizioni relativamente semplici. Ciò rendeva necessario l'introduzione di qualche meccanismo che ne facilitasse la gestione.
Si era pensato come soluzione a dei costrutti per poter scomporre il documento in elementi più piccoli, dei modelli parametrizzati riutilizzabili che avrebbero avuto il vantaggio aggiuntivo di alleggerire l'onere della stesura. Si sarebbe fornito poi un editor per automatizzarne la creazione e la modifica.
La progettazione e l'integrazione di questi strumenti avrebbe d'altra parte inciso inevitabilmente sui tempi di sviluppo.

[2] Cfr.: http://www.w3.org/XML/

Successivamente sono state riscontrate delle incongruenze nelle implementazioni da parte dei linguaggi utilizzati per il *rendering* dei componenti generici. In alcuni casi si sarebbero potuti correggere i comportamenti anomali, in altri le conseguenze si sono rivelate più profonde.
Andavano a crearsi delle discrepanze tra l'albero associato al *DOM* (*Document Object Model*), ossia la gerarchia degli oggetti definiti nell'XML, quello legato ai componenti generici, ottenuti dal parsing dello stesso, e quello dei componenti grafici effettivi.[3]
Le differenti funzionalità conferite ai componenti nei diversi linguaggi, inoltre, variavano considerevolmente. Nel definire il set di componenti generici si era limitati nella scelta di quelle caratteristiche relative al singolo oggetto supportate globalmente. Ciò rendeva molto difficile realizzare interfacce che si presentassero e rispondessero agli input dell'utente in maniera omogenea; risultava compromessa l'effettiva utilità di avere *view* multiple.

In ultimo, l'accesso di basso livello con il quale si permetteva ai `Plugin` di definire la propria interfaccia aveva un effetto collaterale non trascurabile: il componente grafico era scollegato dal proprio comportamento e da quelli dei componenti con i quali avrebbe dovuto interagire.
La dichiarazione di un componente doveva essere accompagnata, in un secondo momento, dalla fornitura alla GUI del relativo controllore. Java in questo caso non era d'aiuto: non mette a disposizione infatti mezzi per collegare il componente a metodi che agiscono su di esso attraverso puntatori a funzione, o *callback function*. Essendo il componente definito per via testuale, l'associazione non poteva che avvenire allo stesso modo. Per quanto fosse possibile raggiungere il risultato desiderato, attraverso ad esempio l'uso di *RMI* (*Remote Method Invocation*), il sistema risultava fragile, basato su corrispondenze tra stringhe.[4]

4.2.3 Ridefinizione degli obiettivi

Plugin e visualizzazione

Alla base delle problematiche riscontrate è proprio la scelta di consentire l'accesso ai plug-in a dettagli a livello di componente grafico. *Ma è davvero una libertà che si desidera dare loro?*

[3] rappresentazione ad oggetti della struttura di un documento, cfr.: http://www.w3.org/DOM/
[4] Cfr.: http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

Sebbene conferisca alla GUI la capacità di rispecchiare in maniera naturale i continui cambiamenti di PariPari, risulta notevole il costo in termini di tempi per lo sviluppo e la manutenzione del codice, difficilmente conciliabili con i *deadline* che il gruppo si era prefisso. Lasciando ai `Plugin` il controllo sulla definizione di porzioni di interfaccia, si lasciava a questi anche la responsabilità di curarne lo stile e l'ergonomia. Non sarebbe stato dunque possibile garantire alcuna uniformità nella visualizzazione e nelle modalità di funzionamento dei vari *widget*, la cui composizione non ne avrebbe potuto rispecchiare la rilevanza per l'utente e le effettive esigenze grafiche.

Rivalutando questa specifica, si può ottenere una struttura più solida e snella, al costo della perdita di una generalità spesso indesiderata.

Portabilità

Allo stesso modo, i vantaggi derivanti dal fornire supporto a *view* multiple possono essere in gran parte conservati attraverso scelte alternative, di più semplice attuazione. La portabilità introdotta da Java permette una potenziale penetrazione nel mercato estremamente ampia. Non vi è motivo di credere che una tale assunzione perda di validità nel breve termine, anzi: tecnologie ad essa associate, come *AJAX* (*Asynchronous JavaScript And XML*), sono in netta espansione.[5]
D'altra parte, proprio le applicazioni *web based* che fanno uso di tali strumenti stanno prendendo piede. L'utente medio è ormai abituato a considerare il *browser* come il portale verso la piattaforma fornita dalla rete, in grado di offrire tutti i servizi più comuni dell'ambiente desktop: ne è palese sintomo lo sviluppo di progetti che vedono convergere i concetti di *browser* e di sistema operativo, come Google Chrome OS.[6]
Realizzando una GUI *web based*, dunque, si può mantenere un'ottima capacità di adattamento ai più diversi dispositivi, potendo sfruttare pienamente le potenzialità del linguaggio scelto.

A tal scopo si è deciso di adottare Vaadin.[7] Vaadin è un *framework* open source per la creazione di RIA (*Rich Internet Applications*), applicazioni web che possiedono le caratteristiche e le funzionalità di quelle desktop.
La sua peculiarità principale è quella di permettere di sviluppare l'interfaccia interamente in Java, utilizzando poi il motore di GWT (*Google Web Toolkit*) per formare la

[5] Cfr.: http://en.wikipedia.org/wiki/Ajax_programming
[6] Cfr.: http://www.chromium.org/chromium-os
[7] http://vaadin.com/home

pagina web in JavaScript altamente ottimizzato. A differenza di GWT, però, in Vaadin la maggior parte della logica viene eseguita lato server, consentendo una maggiore stabilità ed affidabilità.

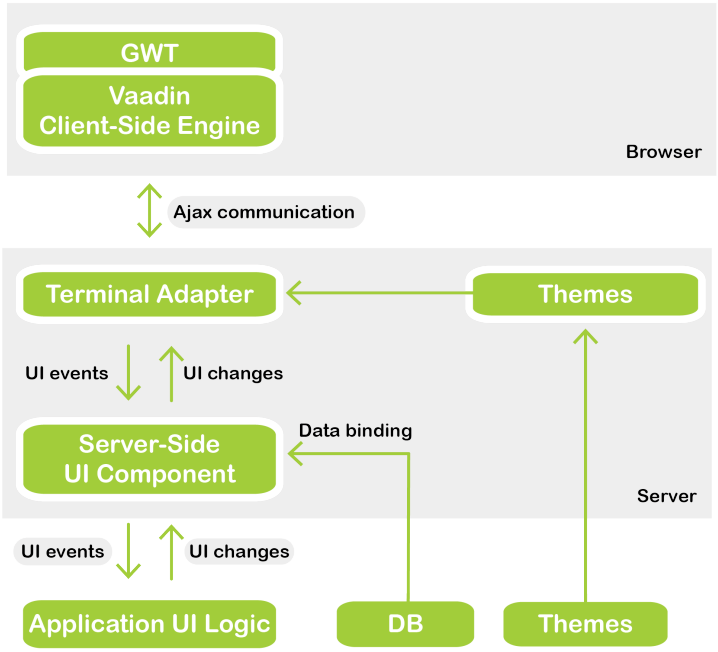

Figura 4.2: *Immagine riassuntiva dell'architettura di Vaadin*

Vaadin offre un ricco insieme di caratteristiche e funzionalità. Ne viene riportata una breve lista, che non vuole essere in alcun modo esaustiva.

- La sicurezza è garantita dalla validazione lato server.
- Il framework comprende un set di componenti completo per tutte le applicazioni più comuni.
- Viene fornito supporto a modelli di programmazione delle interfacce evoluti come event-listener e MCV.
- E' possibile definire temi attraverso CSS (*Cascading Style Sheets*).
- La comunicazione client-server è completamente automatizzata.
- E' supportato da tutti i più diffusi browser.

- L'integrazione con il browser consente un utilizzo coerente delle sue funzionalità, come la navigazione attraverso la cronologia utilizzando i tasti "indietro" e "avanti".
- E' compatibile con tutti i maggiori Application Server e portali Java EE.

La possibilità di programmare in Java consente di scrivere codice facilmente riutilizzabile, e di accorciare i tempi di realizzazione, essendo questo il linguaggio al quale gli sviluppatori di PariPari sono abituati.

4.2.4 Un nuovo modello

Rendere visibili ai `Plugin` informazioni legate alla visualizzazione non è indispensabile, anzi, sotto molti aspetti è sgradito. E' fondamentale per una interfaccia utente dare una rappresentazione delle caratteristiche dell'applicazione tanto completa quanto accessibile: se non è possibile arrivare ai contenuti, perché il percorso per raggiungerli è poco intuitivo o frustrante per l'utente, lo scopo di fornire una GUI risulta inficiato. L'interesse è un presupposto per l'utilità del servizio: perdendo l'uno non si può che perdere l'altro.

Per garantire un'esperienza soddisfacente si deve creare un ambiente con cui l'utente possa relazionarsi facilmente. In primo luogo, bisogna evitare il fenomeno che Leon Festinger ha definito *dissonanza cognitiva*.[8]

Con questo termine si intende la sensazione sgradevole derivante dalla percezione simultanea di due concetti in conflitto o incoerenti. Se l'utente deve compiere due compiti simili tra loro, si aspetta di poterli eseguire in maniera simile, ottenendo simili risultati; se questo non avviene, la conseguenza intuibile è che l'utente si senta sconfortato dal dover assimilare due comportamenti differenti apparentemente senza motivo.

Un esempio negativo illustre in ambito software è il modo in cui in Windows vengono copiati i file attraverso *drag-and-drop*: se si trasferisce un file da una cartella ad un'altra appartenente allo stesso volume logico, il file apparirà nella cartella di destinazione e scomparirà da quella sorgente; se invece le due cartelle si trovano su dischi differenti, non verrà eliminato dalla posizione originale. Il sapere che ciò deriva dall'esecuzione effettiva di due operazioni distinte nei due casi, `MOVE` e `COPY`, rende l'incongruenza tra gli esiti meno oscura, ma non meno fastidiosa.

[8] Cfr.: Leon Festinger, *A theory of cognitive dissonance*, Stanford University Press, 1957.

Si dovrebbe garantire la maggior omogeneità possibile nelle modalità d'interazione tra utente ed applicazione. Proprio a partire da questa idea, si è deciso di spostare l'attenzione dal concetto di componente a quello di *funzionalità*. Alzando in questo modo il livello d'astrazione, adottando la funzionalità come modulo sia nella logica interna di controllo, sia per la comunicazione con i `Plugin`, a fronte di una notevole perdita generalità nella definizione dell'interfaccia, si permette la presentazione coerente delle azioni con cui l'utente può controllare PariPari.

Un'importante conseguenza è che in questo modo è possibile rendere trasparenti ai `Plugin` i dettagli prettamente grafici. Non serve che specifichino *la propria visualizzazione*, ma *le modalità d'intrazione* con l'utente. Si ottiene un'automatica separazione tra i *dati* su cui agiscono `Plugin` ed utente, i *meccanismi* attraverso i quali operare su di essi, e la relativa *rappresentazione*.

4.2.5 Il Plugin GUI

Si può dunque determinare una nuova struttura su cui basare il `Plugin` GUI. Innanzitutto questa dev'essere in grado di risolvere il conflitto generato dalle necessità contrastanti di attenersi alla particolare architettura comunicativa interna a PariPari, illustrata nella sezione 4.1, nella quale lo scambio è anonimo ed episodico, e quella di stabilire una connessione continuativa ed almeno parzialmente conscia delle caratteristiche dell'interlocutore.

E' possibile soddisfare entrambe le esigenze definendo una coppia di interfacce che estendano `paripari.API`, `GUIAPI` e `GraphicalPluginAPI`, e che permettano il controllo reciproco. `GUIAPI` consente ai `Plugin` di comandare la GUI, richiedendo aggiornamenti negli stati dei componenti, o interagendo direttamente con l'utente, per mezzo di notifiche, dialoghi, form. `GraphicalPluginAPI` realizza la comunicazione nell'altro verso, specificando le funzioni messe a disposizione dai `Plugin` all'utente.

La scelta di rappresentare in maniera omogenea funzionalità collegate comporta la necessità di definire un modello per i dati sui quali queste agiscono condiviso. Ciò è in linea con le scelte progettuali fatte in termini di design: all'utente non saranno trasmesse, ad esempio, le differenze negli attributi dipendenti dal protocollo nativo che un contatto possiede, né le informazioni di un file in trasferimento caratteristiche della rete dalla quale proviene.

Similmente, per poter collegare le funzionalità dell'interfaccia ai `Plugin`, i metodi di `GraphicalPluginAPI` che agiscono su dati appartenenti ad aree semantiche affini andranno a costituirne sotto-interfacce specifiche: quando la modifica di dati appartenenti ad una categoria genererà un evento, questo verrà propagato verso tutti i `Plugin` della particolare sotto-interfaccia. In questo modo la GUI è in grado di trasmettere una richiesta a tutti i potenziali interessati, conoscendone solo le informazioni strettamente necessarie a determinarne la natura. Nel caso `Plugin` futuri non rientrassero nelle categorie predeterminate, sarà sufficiente aggiungerne di nuove.

A questo punto, si può comprendere la struttura della GUI seguendo un suo ciclo di vita tipico. All'avvio di PariPari il `Core` carica e lancia il `Plugin` GUI, chiamando il metodo `init()` della classe omonima. Dopo aver ultimato le debite inizializzazioni, e creato la cartella base in cui salvare i file di *log*, la classe `GUI` si occupa di avviare il thread `Listener`, che si mette in ascolto delle richieste di `GUIAPI` provenienti dal `Core` da parte degli altri `Plugin`. Per risposta verrà restituito un nuovo esemplare di `GUIAPIImpl`, un oggetto che realizza `GUIAPI`. Ogni richiesta dovrà avere come parametro in ingresso il controllore grafico del `Plugin`, ossia la classe che fornisca i metodi di `GraphicalPluginAPI` e delle eventuali sotto-interfacce. Un gestore si occuperà di mantenere aggiornata una tabella contenente le `GraphicalPluginAPI` dei `Plugin` connessi alla GUI nella particolare sessione, suddivise per categoria. Tutte le richieste successive alla prima, al di là di quella di disconnessione, avverranno dunque senza il tramite del `Core`, direttamente attraverso `GUIAPI`, velocizzando la comunicazione nel caso venisse richiesto un numero elevato di operazioni.

Poiché la GUI stessa è un plug-in, è possibile inibirne il caricamento per controllare PariPari attraverso la `Console` fornita dal `Core`.

Sempre durante l'esecuzione del metodo `init()` di `GUI`, per mezzo della classe `ServerStarter` viene avviato un server e viene eseguito a *runtime* il *deployment* del file *WAR* (*Web application archive*, formato utilizzato per creare archivi di applicazioni web) contenente il *servlet*, responsabile effettivo della gestione della parte grafica.[9] In questo modo si potrà facilmente passare da una GUI locale ad una remota, senza necessità di introdurre ulteriori *layer* nella logica di controllo.

La comunicazione tra plug-in e servlet avviene grazie alla classe `GUIThread`, attivata anch'essa da `GUI`, e dalla controparte `ViewThread` lato servlet. `GUIThread` crea

[9] Cfr.: http://java.sun.com/developer/technicalArticles/Servlets/servletapi/

un `ServerSocket`, ed attende che `ViewThread` richieda una connessione su una porta determinata *a priori*. In caso di connessione avvenuta con successo, attiva due ulteriori thread, `InputReader` ed `OutputWriter`.
Le informazioni da scambiare vengono codificate in richieste specifiche, ed incapsulate in oggetti secondo l'interfaccia `ICommand`. Per il trasferimento dei comandi si fa uso della serializzazione, in modo che il ricevente possa ricostruire gli oggetti inviati dal mittente. [10]

Per garantire una comunicazione il più possibile asincrona, per mantenere un alto livello di interattività con l'utente, una volta modellate le richieste provenienti dai `Plugin` in `ICommand`, questi vengono aggiunti in una coda di comandi uscenti.
La funzione di `OutputWriter` è quella di inviare questi ultimi sul *ObjectOutputStream*. In maniera duale, *InputReader* è incaricato di leggere i comandi provenienti da `ViewThread` e di inserirli in una coda di comandi in ingresso. `GUIThread` attende che un nuovo `ICommand` risulti disponibile, estrae ed interpreta la richiesta in esso contenuta, e la inoltra eventualmente ai `Plugin`.

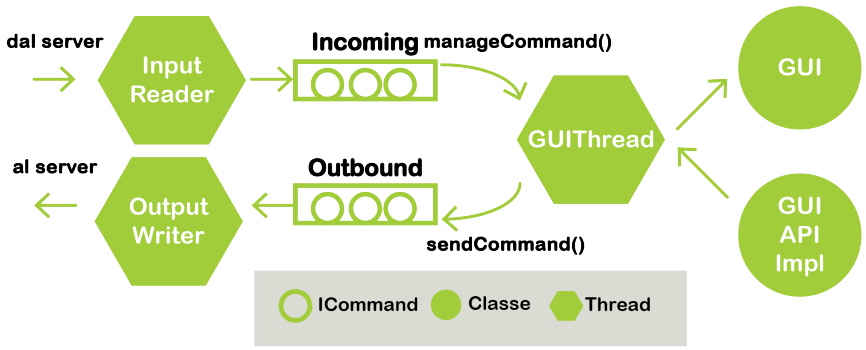

Figura 4.3: *Schema del meccanismo di comunicazione interna tra plug-in GUI e web application, visto dal lato del plug-in*

[10] Cfr.: http://java.sun.com/developer/technicalArticles/Programming/serialization/

Per garantire solidità ed efficienza nella sincronizzazione si è scelto di utilizzare come realizzazione delle code *incoming* ed *outbound* delle `LinkedBlockingQueue`. Nel caso un comando necessiti di una risposta, come avviene ad esempio se si ha bisogno di una conferma da parte dell'utente, il thread richiedente viene addormentato finché questa non venga fornita.

Il funzionamento dalla parte del servlet è simmetrico. L'applicazione web viene avviata quando viene aperto il browser all'*URL* corrispondente. `ViewThread` attraverso un `Socket` tenta di connettersi al `Plugin`, fa partire i propri thread lettore e scrittore, ed attende l'arrivo di comandi da gestire.

All'interno del servlet si è deciso di separare il *data model* dalla *view* in senso stretto: in questo modo è possibile ottenere più rappresentazioni coerenti di uno stesso dato e separare in maniera più efficace la logica applicativa dalla visualizzazione.

Avendo optato per l'utilizzo di Vaadin per la realizzazione del servlet, ci si è avvalsi degli strumenti messi a disposizione dal linguaggio per vincolare i componenti grafici ad un modello di dati.[11] Tali strutture sono composte dalle interfacce nel package `com.vaadin.data`, si fa riferimento in particolare a `Property`, `Item` e `Container`. Come analogia, il rapporto di gerarchia tra tali elementi potrebbe essere assimilato a quello che intercorre tra una cella, una riga ed una tabella in un foglio di calcolo. `Property` rappresenta un dato generico, `Item` una lista di differenti `Property`, `Container` è una collezione di `Item`. Le definizioni del *data model* proposte permettono di notificare alle visualizzazioni i cambiamenti avvenuti attraverso l'invio di eventi.

Quando `ViewThread` riceve comandi che modificano lo stato delle risorse rappresentate, vengono richiamate le funzioni corrispondenti della classe `Resource Warehouse`, il gestore del *data model*. A questo è sincronizzato `ResourceViewer`, responsabile della creazione e del mantenimento delle componenti grafiche.

4.3 Considerazioni sulla realizzazione

In questo capitolo si è presentata brevemente la struttura del plug-in GUI, e si sono discusse le difficoltà incontrate nella progettazione e le motivazioni alle scelte fatte. E' prevedibile che al momento dell'estensione delle funzionalità, e dal confronto con

[11] http://vaadin.com/book/-/page/datamodel.html

le esigenze degli altri plug-in, sorgano problemi che possano comportare modifiche e adattamenti in corso d'opera. Ci si augura che il lavoro svolto sia servito a porre delle basi sufficientemente solide per un contributo al progetto duraturo.

Conclusioni

Si è visto come la progettazione dell'interfaccia grafica per PariPari abbia richiesto competenze aggiuntive rispetto a quelle generalmente richieste per lo sviluppo di software.
Innanzitutto è stato necessario comprendere la posizione che PariPari assumerà nel mercato e quali siano le sue caratteristiche peculiari in confronto alle piattaforme concorrenti, per poterne valorizzare gli aspetti innovativi.
Si sono poi dovute analizzare le esigenze e gli obiettivi degli utenti finali, allo scopo di definire ed ottimizzare le modalità d'interazione con l'applicazione.
Lo studio dell'ergonomia e della veste grafica fin nel dettaglio è stato lungo ed impegnativo: in particolare, la difficoltà principale che si è incontrata è stata l'individuazione di elementi in grado di garantire una rappresentazione intuitiva ed allo stesso tempo versatile della multifunzionalità di PariPari.
Le problematiche legate alla realizzazione della GUI come plug-in, in special modo quelle relative all'integrazione con la piattaforma, hanno reso indispensabile effettuare una verifica dei requisiti ed un'attenta progettazione.
Si può prevedere che, al momento del lancio di PariPari, la risposta da parte degli utenti renderà necessarie ulteriori modifiche per il perfezionamento dell'usabilità dell'interfaccia. Con questa tesi si spera di aver contribuito a definire i principi chiave e le linee guida per lo sviluppo di una GUI che possa andare incontro ai bisogni degli utenti e determinare, o quantomeno favorire, il successo di PariPari.

Appendice A

Codice

In questa sezione vengono riportati degli estratti dal codice che compone il plug-in GUI nello stato attuale. Per quanto possano essere soggetti a modifiche in futuro, sono degli utili riferimenti per meglio comprendere il *framework* per l'interfacciamento con PariPari esposto nel capitolo 4.

A.1 GUIAPI

```
1  public interface GUIAPI extends API {
2
3    /**
4     * Adds a new contact to the contact list.
5     *
6     * @param newContact
7     *            the new contact
8     */
9    public void addContact(GUIContact newContact);
10
11   /**
12    * Adds an array of new contacts to the contact list.
13    *
14    * @param newContacts
15    *            the new contacts
16    */
17   public void addContacts(GUIContact[] newContacts);
18
19   /**
20    * Refreshes the status of a contact.
21    *
22    * @param contact
23    *            the contact
24    */
25   public void refreshContact(GUIContact contact);
26
27   /**
28    * Posts an incoming message to the specified conversation.
```

```
29      *
        * @param message
31      *            the message
        * @param conversation
33      *            the conversation
        */
35      public void postIncomingMessage(GUIMessage message,
            GUIConversation conversation);
37
        /**
39      * Refreshes the status of a shared file.
        *
41      * @param sharedFile
        *            the shared file
43      */
        public void refreshSharedFile(GUISharedFile sharedFile);
45
        /**
47      * Shows the user a notification.
        *
49      * @param message
        *            the notification message
51      */
        public void notifyUser(String message);
53
        /**
55      * Asks user to make a yes/no choice. This is a blocking method, as it waits
        * for the user to answer.
57      *
        * @param question
59      *            the question
        * @return 0 if user chose yes, 1 if no, -1 if an error occurred
61      */
        public int askUser(String question);
63
        /**
65      * Asks user to make a choice between the provided options. This is a
        * blocking method, as it waits for the user to answer.
67      *
        * @param question
69      *            the question
        * @param options
71      *            the available options
        * @return the index of the option the user chose, -1 if an error occurred,
73      *         or if the user did't make a choice
        */
75      public int askUser(String question, String[] options);

77      /**
        * Asks the user to choose a file. This is a blocking method, as it waits
79      * for the user to answer.
        *
81      * @param message
        *            the message to display
83      * @return the file the user chose, null if an error occurred, or if the
        *         user didn't make a choice
```

```
85      */
        public File askUserFileChoice(String message);
87
        /**
89       * Asks the user to choose a file. This is a blocking method, as it waits
         * for the user to answer.
91       *
         * @param message
93       *            the message to display
         * @param filter
95       *            the file name filter
         * @return the file the user chose, null if an error occurred, or if the
97       *         user didn't make a choice
         */
99      public File askUserFileChoice(String message, FilenameFilter filter);

101     /**
         * Asks the user to choose a file. This is a blocking method, as it waits
103      * for the user to answer.
         *
105      * @param message
         *            the message to display
107      * @param rootDirectory
         *            the root directory for the user to choose from
109      * @return the file the user chose, null if an error occurred, or if the
         *         user didn't make a choice
111      */
        public File askUserFileChoice(String message, String rootDirectory);
113
        /**
115      * Asks the user to choose a file. This is a blocking method, as it waits
         * for the user to answer.
117      *
         * @param message
119      *            the message to display
         * @param rootDirectory
121      *            the root directory for the user to choose from
         * @param filter
123      *            the file name filter
         * @return the file the user chose, null if an error occurred, or if the
125      *         user didn't make a choice
         */
127     public File askUserFileChoice(String message, String rootDirectory,
            FilenameFilter filter);
129
        /**
131      * Shows a dialog containing the information about an undergoing task.
         *
133      * @param newTask
         *            the task
135      */
        public void showTask(GUITask newTask);
137
        /**
139      * Refreshes the status of the task.
         *
```

```
141     * @param task
        *            the task
143     */
    public void refreshTask(GUITask task);
145
    /**
147     * Removes the task dialog.
        *
149     * @param task
        *            the task
151     */
    public void removeTask(GUITask task);
153
    /**
155     * Prompts user for login.
        *
157     * @param message
        *            the message to display
159     * @return an array with the username and password the user submitted, null
        *         if an error occurred or if the user didn't submit anything
161     */
    public String[] askLogin(String message);
163
}
```

GUIAPI.java

Attraverso l'interfaccia `GUIAPI` i `Plugin` possono controllare la propria porzione di GUI. I suoi metodi consentono di aggiornare lo stato dei componenti grafici senza fare direttamente riferimento ad essi; questo è reso possibile dalla separazione della visualizzazione in senso stretto dal modello dei dati da rappresentare. I `Plugin` agiscono infatti su oggetti, quali sono `GUIContact`, `GUIMessage`, `GUIConversation`, `GUISharedFile` e `GUITask`, che appartengono al *data model*, e che nascondono quindi ogni dettaglio di natura grafica. Ciò permette di modificare la realizzazione dell'interfaccia utente, o addirittura di sostituire per intero il modulo GUI, in maniera trasparente al resto di PariPari.

A.2 ICommand

```
public interface ICommand extends Serializable {
2
    /**
4     * The Type of the command.
      */
6   public enum Type {

8     STOP("stop"), PRINT("print"), ASK_USER("ask_user"), ASK_USER_LOGIN(
          "ask_user_login"), ASK_USER_FILE_CHOICE("ask_user_file_choice"), CONTACT_UPDATE(
10        "contact_update"), INCOMING_MESSAGE("incoming_message"), NOTIFY(
```

```
        "notify"), REPLY("reply"), TASK("task");

    /** The string representation of the type. */
    private String type;

    /**
     * Instantiates a new command type.
     *
     * @param type
     *            the type
     */
    Type(String type) {
      this.type = type;
    }

    /**
     * Gets the type of the command.
     *
     * @return the type
     */
    public String getType() {
      return type;
    }
  }

  /**
   * Tells if this command needs an answer.
   *
   * @return true, if it does
   */
  public boolean needsAnAnswer();

  /**
   * Checks if the command is an answer.
   *
   * @return true, if it is an answer
   */
  public boolean isAnAnswer();

  /**
   * Gets the command ID.
   *
   * @return the ID
   */
  public long getID();

  /**
   * Gets the command type.
   *
   * @return the type
   */
  public Type getType();

  /**
   * Gets the command value.
   *
```

```
     * @return the value
68   */
    public Serializable getValue();
70
    /**
72   * Sets the command value.
     *
74   * @param value
     *            the new value
76   */
    public void setValue(Serializable value);
78
}
```

ICommand.java

L'interfaccia ICommand permette lo scambio di informazioni tra il plug-in GUI ed il servlet. I comandi da trasmettere vengono infatti codificati in richieste specifiche ed aggiunti al *payload* di `ICommand`; questi vengono poi serializzati e trasferiti attraverso socket. Una volta ricostruito l'oggetto ICommand, è possibile recuperare la richiesta e determinarne il tipo grazie ai metodi `getValue()` e `getType()`.

Bibliografia

[1] Paolo Bertasi, *Progettazione e realizzazione in Java di una rete peer to peer anonima e multifunzionale*, Dipartimento di Ingegneria dell'Informazione, Università di Padova, 2004.

[2] Tim O'Reilly, *What Is Web 2.0 - Design Patterns and Business Models for the Next Generation of Software*, 30 Settembre 2005.

[3] John Musser, Tim O'Reilly & O'Reilly Radar Team, *Web 2.0 Princiles and Best Practices*, Novembre 2006.

[4] Simone Vidotto, *PariGUI 2009*, Dipartimento di Ingegneria dell'Informazione, Università di Padova, 2009.

[5] Alan Cooper, Robert Reimann, David Cronin, *About face 3: the essentials of interaction design*, Wiley, 2007.

[6] Wilbert O. Galitz, *The Essential Guide to User Interface Design - An Introduction to GUI Design Principles and Techniques*, Wiley, 2007.

[7] Donald Norman, *The design of everyday things*, Basic Books, 1988.

Elenco delle figure