POLITECNICO DI MILANO
Facoltà di Ingegneria - Dipartimento di Elettronica e Informazione

---

Progetto di Laboratorio software

Facebook Automatic List Suggestion

Studenti
Bessi Marco [736443]
Bruni Leonardo [736444]

Curatori
Ing. Davide Eynard
Ing. David Laniado
Ing. Riccardo Tasso

Anno Accademico 2009-2010

Indice

1 Introduzione **1**

2 Architettura **3**

3 Implementazione **6**
3.1 L'applicazione web 6
3.2 Cluster 10
3.3 AccettaLista 14

4 La base di dati **16**

5 Conclusioni e sviluppi futuri **21**

A Definizioni dei formati **23**

Acronimi **27**

Bibliografia **28**

Elenco delle figure

1.1 Esempio di una rete di amicizie visualizzata come un grafo. . . 2

2.1 Architettura. 4

3.1 I package e le classi dell' *applicazione web*. 7
3.2 Le pagine jsp dell' *applicazione web*. 8
3.3 Struttura della visualizzazione di un cluster. 9
3.4 Autocompletamento nell'inserimento di un nuovo amico alla lista. 9
3.5 I package e le classi dell'applicazione *Cluster*. 11
3.6 I package e le classi dell'applicazione *AccettaLista*. 14

4.1 Modello ER della base di dati. 16

A.1 Esempio del formato definito per i risultati degli algoritmi di clustering. 24
A.2 Esempio del formato definito per i risultati da inviare all'applicazione web. 24
A.3 Esempio del formato per i risultati come ricevuto dall'applicazione web. 25
A.4 Esempio del formato definito per l'esportazione dei risultati. . 26

1 Introduzione

I Social Network Sites (SNSs) quali ad esempio MySpace[1] e Facebook[2] permettono agli utenti di presentarsi, gestire le proprie reti di conoscenze e stabilire o mantenere contatti con altre persone. I partecipanti possono usare questi siti per interagire con altre persone conosciute offline oppure conoscerne di nuove. L'applicazione sviluppata in questo progetto è relativa a Facebook, il quale permette ai propri utenti di creare un proprio profilo, mettersi in relazione con amici che possono scrivere commenti sulle relative pagine, vedere i profili di altri utenti e unirsi a gruppi virtuali basati su interessi comuni. Esiste inoltre un secondo strumento che permette di raggruppare gli amici basandosi su qualche criterio che li accomuna: le *liste*. Una lista è infatti un elenco di amici identificato da un nome che in qualche modo rappresenta un collegamento tra tutti loro, ad esempio il fatto di essere compagni universitari piuttosto che amici del mare. Le liste sono utili per l'invio di messaggi mirati oppure per gestire in maniera migliore le proprie politiche di privacy relativamente ai contenuti che si decide di rendere disponibili sul social network. Obiettivo del progetto è quello di sviluppare un'applicazione Facebook che analizzi la rete di amicizie di un utente e proponga in maniera automatica delle possibili liste di amici.

Intuitivamente, la rete di amicizie di un utente verrà convertita in un grafo, composto da tanti nodi quanti sono i suoi amici e da tanti archi quanti sono i rapporti di amicizia che li legano. In altre parole ogni nodo del grafo rappresenterà un amico dell'utente che esegue l'applicazione, mentre ogni arco indicherà il legame di amicizia tra due utenti (ovvero il fatto che i due si trovano reciprocamente nella lista di amicizie di Facebook dell'altro). In figura 1.1 viene mostrata una possibile visualizzazione di tale grafo così come

[1] `http://www.myspace.com/`
[2] `http://www.facebook.com/`

creato dall'applicazione SocialGraph[3] già presente su Facebook. L'applicazione citata, così come tutte le altre ad essa simili, si limita tuttavia alla semplice visualizzazione della rete di amicizie. Anche solo guardando il grafo ci si rende però conto del fatto che ci sono dei cluster, quindi è pensabile applicare algoritmi già noti detti di Community Detection.

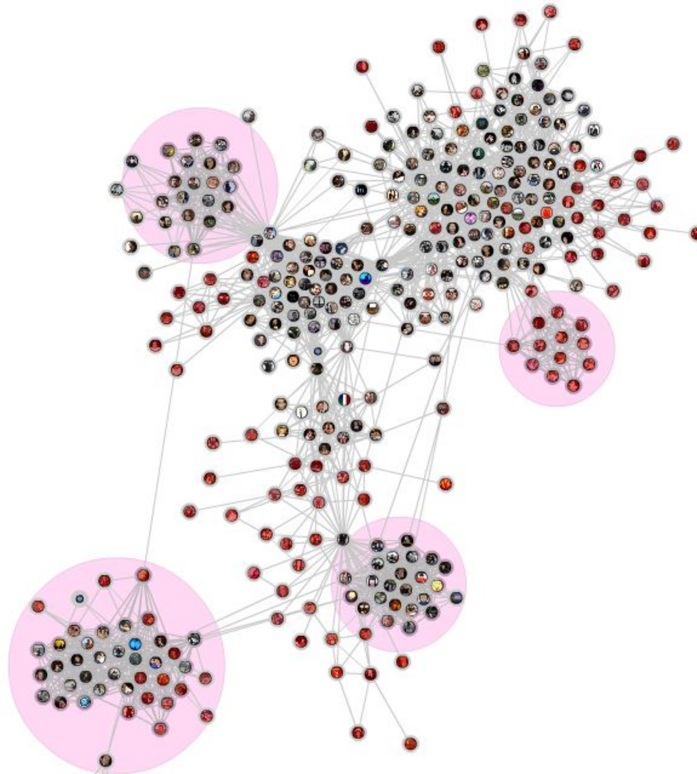

Figura 1.1. Esempio di una rete di amicizie visualizzata come un grafo.

Nei capitoli seguenti verrà dapprima illustrata l'architettura dell'applicazione sviluppata, evidenziando le interazioni tra i vari componenti software coinvolti (cap. 2). Tali componenti verranno discussi in dettaglio nel capitolo 3. Data l'importanza che la base di dati assume per l'esecuzione dell'applicazione sviluppata, il capitolo 4 sarà dedicato a illustrare in maniera dettagliata la sua struttura. Infine in appendice A verranno mostrati i formati definiti per permettere lo scambio di informazioni tra i vari componenti.

[3]`http://apps.facebook.com/socgraph/`

2 Architettura

L'approccio alla base di tutto il processo di sviluppo è stato incentrato sulla possibilità di poter ampliare dinamicamente il numero di algoritmi di clustering utilizzati, senza cioè dover modificare il codice dell'applicazione. Inoltre, benché l'applicazione web sia stata sviluppata in linguaggio Java, non si voleva vincolare anche gli algoritmi di clustering a tale linguaggio. L'obiettivo era infatti quello di poter utilizzare anche algoritmi scritti in linguaggi più efficienti.
Ulteriore obiettivo del progetto era quello di poter analizzare e confrontare la bontà di diversi algoritmi di clustering. Per raggiungere tale obiettivo si è previsto di realizzare una base di dati nella quale raccogliere informazioni di ogni singola esecuzione di ciascun algoritmo.
La soluzione adottata è stata quindi quella di mantenere su Web Server[1] (da adesso indicato come *Javellotto*) unicamente la parte di presentazione e interazione con l'utente, mentre tutta la logica applicativa, nonché la base di dati, è stata progettata per essere disaccoppiata dalla parte di presentazione. Si è quindi pensato di utilizzare una macchina[2] (da adesso indicato come *Jobe*) con elevate capacità elaborative. In particolare su quest'ultima sono stati sviluppati due moduli (memorizzati su Jobe come JAR eseguibili) atti a eseguire dapprima l'elaborazione delle richieste di clustering e in seguito ad aggiornare il database qualora l'utente decida di salvare alcune delle liste proposte (eventualmente da lui modificate). Sempre su Jobe saranno memorizzati i vari algoritmi di clustering sviluppati. L'architettura appena descritta è illustrata in figura 2.1.

L'esecuzione avviene attraverso le seguenti macro-fasi, dettagliate nel capitolo 3:

[1] `http://javellotto.ws.dei.polimi.it/`
[2] `jobe.elet.polimi.it`

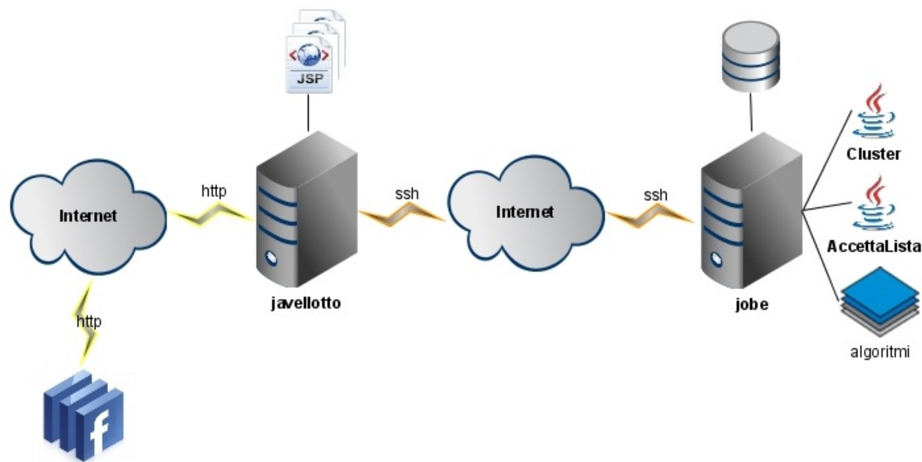

Figura 2.1. Architettura.

1. L'utente, accedendo all'applicazione dal proprio account Facebook, avvia l'esecuzione.

2. L'applicazione web recupera i dati relativi alla rete di amicizie dell'utente, crea un grafo che rappresenta esattamente tali rapporti e invia a Jobe una serializzazione della rete in formato Pajek [1] tramite il comando SCP. Successivamente invoca il modulo *Cluster* che si occuperà di applicare uno degli algoritmi.

3. Il modulo Cluster interroga il database al fine di selezionare in maniera casuale[3] uno degli algoritmi di clustering implementati. Invoca quindi il processo esterno che eseguirà l'algoritmo selezionato e ne attende i risultati rappresentati come stringa XML.

4. L'applicazione Cluster, una volta ripreso il controllo dell'esecuzione, salva su database i dati significativi dell'esecuzione e le informazioni sui singoli cluster restituiti dall'algoritmo appena eseguito. Infine restituisce i risultati all'applicazione web come stringa XML.

5. L'applicazione web propone all'utente le liste risultanti dall'esecuzione dell'algoritmo di clustering e per ciascuna di esse permette eventualmente l'aggiunta o la rimozione di amici.

[3]Si vuole sottolineare il fatto che allo stato attuale la selezione viene effettuata in maniera casuale, ma in futuro si potrà decidere di eseguire un algoritmo piuttosto che un altro in base a dati relativi alla tipologia di rete (es: numero elevato di nodi) oppure a performance di esecuzioni precedenti di un certo algoritmo.

6. Per ognuna delle liste che l'utente ha deciso di salvare, l'applicazione web effettua una invocazione SSH all'applicazione *AccettaLista* richiedendo la modifica della tupla relativa a quella lista, al fine di tenere traccia del fatto che è stato scelto di salvarla e di quanti utenti siano stati aggiunti/rimossi dalla lista proposta.

7. L'applicazione AccettaLista esegue la modifica del database e restituisce il controllo al chiamante.

8. Infine l'applicazione web mostra all'utente una pagina riepilogativa delle liste salvate e permette di esportarle in formato XML.

È interessante sottolineare come questo tipo di architettura realizzi gli obiettivi esposti in apertura a questo capitolo. In particolare l'unico obiettivo che necessita di ulteriori commenti è quello relativo alla possibilità di aggiungere dinamicamente nuovi algoritmi di clustering, in quanto gli altri obiettivi risultano raggiunti in maniera ovvia da quanto già esposto.

Componente chiave che permette di raggiungere quanto prefissato è il database, il quale verrà ulteriormente descritto nel capitolo 4. Nel momento in cui si vuole aggiungere un nuovo algoritmo, sarà sufficiente indicarne la "presenza" inserendo una tupla nella base di dati che conterrà il comando che occorre invocare per eseguirlo. Nel caso in cui esso necessiti di alcuni parametri in input per essere eseguito, è stata prevista la possibilità di inserire apposite tuple che permettono proprio di specificare quali e quanti saranno tali parametri. In questo modo, il modulo Cluster, nel momento in cui selezionerà un algoritmo per mezzo di interrogazioni al database, avrà a disposizione tutte le informazioni disponibili su come poterlo invocare.

Come ultima osservazione generale, occorre far notare che su Jobe non vengono mai effettuati salvataggi di file durante l'intera elaborazione, ma i passaggi di informazione tra i vari componenti avvengono esclusivamente mediante i buffer di Input e Output. Questo per evitare che in caso di crash di un qualsiasi componente rimangano salvati dei file sul server. Discorso leggermente differente va fatto per il Web Server, in quanto il file in formato Pajek della rete deve essere creato per permetterne il passaggio tramite SCP all'applicazione Cluster residente su Jobe. Tale file sarà eliminato al momento della ricezione dei risultati, o per meglio dire al ritorno del controllo dopo l'invocazione SCP, in quanto anche in caso di un crash di applicazioni su Jobe, tale invocazione comunque renderà il controllo al chiamante.

3 Implementazione

Nel presente capitolo viene affrontata in maniera dettagliata l'implementazione di ogni singolo componente software del progetto. In particolare sono stati implementati tre componenti: il *primo*, realizzato come un'applicazione web, che ha lo scopo di gestire l'interazione con l'utente (sez. 3.1); il *secondo* relativo all'esecuzione dell'algoritmo di clustering (sez. 3.2); il *terzo* di aggiornamento del database in seguito al salvataggio di una lista da parte dell'utente (sez. 3.3). A supporto dell'applicazione è stata progettata una base di dati che verrà discussa nel capitolo 4.

3.1 L'applicazione web

L'applicazione web gestisce e visualizza l'interfaccia grafica all'utente all'interno di Facebook. In figura 3.1 è mostrata la struttura delle classi *Java*.

Nel package `facebook` sono contenute le classi di utilità per gestire l'interazione con i servizi di Facebook, come la gestione dell'autenticazione e le classi di gestione dei legami di amicizia e degli utenti. Inoltre contiene la classe `Gestione` che abilita tutte le operazioni di clustering.
Il package `settings` contiene la classe adibita a contenitore delle stringhe di accesso ai servizi utilizzati dall'applicativo, quali database o connessioni remote.
Il package `utils` contiene la classe che permette di generare dinamicamente un file XML su richiesta dell'utente nel momento in cui decida di esportare i risultati.
Il package `ssh` contiene tutte le classi che realizzano il collegamento tramite SSH tra l'applicazione web e i componenti software ospitati sul server Jobe.

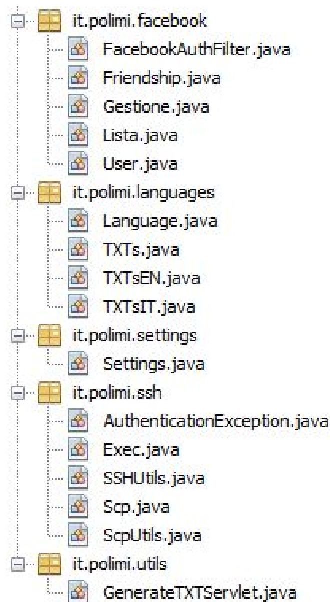

Figura 3.1. I package e le classi dell' *applicazione web*.

In particolare le operazioni implementate sono relative all'invocazione di servizi remoti e al trasferimento di file.
Il package `languages` gestisce l'internazionalizzazione dell'applicazione. Al momento sono state implementate la lingua italiana e inglese. Nel momento in cui si abbia la necessità di aggiungere una nuova lingua è sufficiente effettuare una nuova implementazione dell'interfaccia che espone i metodi relativi a tutte le stringhe testuali che sono utilizzate nell'interfaccia grafica.

Per quanto riguarda l'interfaccia grafica vera e propria sono state realizzate tre pagine JSP che hanno il compito di guidare l'utente nell'utilizzo dell'applicazione. In figura 3.2 è mostrata la struttura delle pagine JSP.
Un'applicazione Facebook deve essere prima di tutto creata tramite l'applicazione ufficiale *Developer* [2] di Facebook. Tramite questa applicazione è possibile impostare varie proprietà dell'applicazione che andremo a creare. Ognuna di queste proprietà è modificabile dinamicamente anche a progetto creato, anche se questo potrebbe risultare dannoso. Ad esempio, una di queste proprietà è *Metodo di restituzione* che permette di impostare se le pagine dell'applicazione verranno renderizzate esclusivamente con i tag FBML o con i tag XFBML/HTML (selezionando *iFrame*); iniziato lo sviluppo dell'applicazione, modificare questa proprietà comporterebbe riscrivere totalmente le pagine che altrimenti non sarebbero più renderizzate da facebook. In una

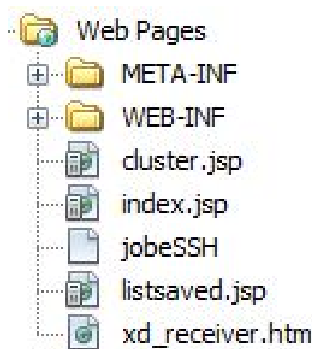

Figura 3.2. Le pagine jsp dell' *applicazione web.*

applicazione FBML i tag utilizzati sono esclusivamente FBML e per inserire codice html si deve usare un tag FBML apposito che ne permette il rendering. Al contrario, una applicazione *iFrame* è scritta in HTML standard e non è permesso usare tag FBML ma solo tag XFBML (che sono in minor numero e ricoprono solo un numero limitato di funzioni); per permettere l'uso di tag FBML in una applicazione *iFrame* esiste un tag XFBML apposito al cui interno vanno inseriti in cascata un tag `<script>` e il tag FBML che vogliamo renderizzare. Quindi entrambe le opzioni hanno dei pro e dei contro che devono essere valutati attentamente per effettuare la scelta più opportuna. Nel nostro caso la scelta è ricaduta su *iFrame*. Il motivo è dovuto al fatto che l'applicazione utilizza in ampia parte contenuto HTML, questo avrebbe reso necessario l'intenso uso di tag FBML capaci di visualizzarlo correttamente con notevole decadimento delle prestazioni di renderizzazione.

Nella pagina di presentazione (`index.jsp`) viene illustrato lo scopo dell'applicazione e il suo funzionamento, oltre alla possibilità di modificare la lingua dell'applicazione. All'interno della prima pagina viene creato l'oggetto `Gestione`, unico per ogni singola esecuzione, e per questo memorizzato nella sessione dell'Application Server in modo da poter essere recuperato dalle altre pagine JSP. Con un pulsante è quindi possibile lanciare l'applicazione vera e propria che andrà ad analizzare la lista di amicizie. Nella pagina successiva (`cluster.jsp`) è situata la chiamata al metodo di clustering. Come prima cosa la pagina si occuperà di recuperare l'oggetto `Gestione` dalla `session`, quindi, tramite l'apposita funzione `gestione.cluster()` farà partire secondo un certo criterio (attualmente in modo casuale) uno degli algoritmi sviluppati. Come risultato del metodo viene restituito un oggetto di tipo `HashMap` contenente la lista dei cluster trovati, ciascuno associato all'id della tupla del database cui fa riferimento. Di seguito viene creata la struttura

dati contenente tutti gli amici dell'utente che sarà poi utilizzata per l'autocompletamento dei nomi durante l'inserimento di un nuovo amico in una determinata lista. Nella pagina vengono quindi resi visibili i dati interessanti generati dall'algoritmo (come tempo di esecuzione o nome dell'algoritmo utilizzato), una breve spiegazione dei risultati, delle istruzioni su come procedere con l'uso dell'applicazione e infine i cluster individuati dall'algoritmo. Ogni singolo cluster è visualizzato come in figura 3.3. Nella prima parte del

Figura 3.3. Struttura della visualizzazione di un cluster.

riquadro viene riportato un numero progressivo con a fianco la possibilità di inserire un nome di riconoscimento per la lista e la checkbox per abilitare il salvataggio della stessa. Di seguito viene poi visualizzato un elenco di utenti, ciascuno dei quali con la propria checkbox, da deselezionare se non si vuole l'utente all'interno della lista corrente (di default è selezionato), la propria immagine e il proprio nome Facebook. Nella parte finale si ha la colonna per l'inserimento di nuovi amici all'interno della lista. Basterà iniziare a scrivere il nome di un amico affinché l'autocompletamento entri in funzione per suggerire gli amici che hanno all'interno del proprio nome Facebook la stringa digitata (fig. 3.4). Per la funzione di autocompletamento sono stati usati i javascript di *jquery* [3]. Purtroppo non è stato possibile utilizzare il tag

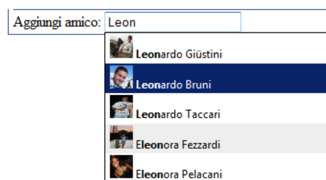

Figura 3.4. Autocompletamento nell'inserimento di un nuovo amico alla lista.

di Facebook appropriato perché non esiste in XFBML e quindi sarebbe sta-

to necessario usare l'apposito tag FBML. Come spiegato in precedenza però questo comporta l'utilizzo del tag `<script>` e ciò rende impossibile il riutilizzo dei valori inseriti, rendendolo quindi inutilizzabile per la nostra necessità. Alla fine della pagina è presente il pulsante "salva liste" che permette, dopo aver effettuato tutte le modifiche necessarie e aver selezionato quelle che interessa salvare, di passare i dati alla pagina successiva (`listsaved.jsp`). In quest'ultima pagina JSP viene per prima cosa recuperato l'oggetto `Gestione` e dopo una breve spiegazione vengono mostrate le liste create precedentemente (modificate secondo le necessità dell'utente). L'idea iniziale era di salvare direttamente le liste all'interno del profilo dell'utente in modo da renderle direttamente utilizzabili. In questo modo l'applicazione sarebbe risultata più utile e immediata, tuttavia non sono presenti metodi nelle API messe a disposizione da Facebook che permettano questa funzione. Per far fronte a questa limitazione attuale (eventualmente correggibile qualora in un futuro aggiornamento delle API di Facebook venga resa disponibile la funzionalità necessaria) l'idea si è quindi spostata sullo sviluppo di uno script esterno all'applicazione, sviluppato dai tutor del progetto, che permettesse in maniera automatica il loro salvataggio. In questo caso il problema di usabilità è relativo alla segretezza della login e della password Facebook. Infatti per il funzionamento dello script è necessario inserire tali dati sensibili. Rimane quindi a discrezione dell'utente la possibilità di utilizzare o meno questa funzionalità. Il codice XML da passare allo script viene generato automaticamente dall'applicazione e reso disponibile da scaricare all'utente tramite un link su un'immagine all'interno dell'ultima pagina JSP, all'altezza della visualizzazione delle liste salvate.

Infine viene posta attenzione al file *xd_receiver.htm* il quale richiama un javascript fornito da Facebook che permette l'adattamento dinamico e automatico dell'altezza dell'applicazione oltre che il rendering dei tag XFBML.

3.2 Cluster

Scopo principale di questo modulo è quello di gestire l'esecuzione del clustering e di rendere disponibili i risultati all'applicazione web. In figura 3.5 ne è mostrata la struttura.

Nel package `database` sono contenute classi di utilità per gestire l'interazione con il database. Sono resi disponibili metodi per effettuare sia la connessione che le operazioni di manipolazione di base tramite SQL.
Il package `settings` contiene la classe adibita a contenitore delle impostazioni dell'applicazione. In particolare sono contenuti eventuali path, dati di accesso e URL dei servizi cui è necessario connettersi.

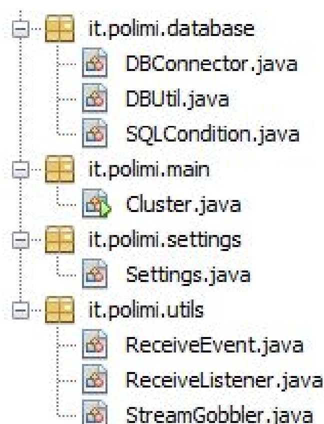

Figura 3.5. I package e le classi dell'applicazione *Cluster*.

Il package `utils` contiene classi che risolvono dei problemi relativi all'invocazione della funzione di libreria Java `Runtime.exec()`. Tali problemi verranno affrontati in dettaglio nel corso di questa sezione.
Il package `main` contiene la classe principale che sarà eseguita in seguito all'invocazione.

Il modulo è reso disponibile su Jobe come file JAR eseguibile ed è invocabile per mezzo del comando `java -jar ./apps/Cluster/Cluster.jar netFileName [name]`, dove *netFileName* è il nome del file in formato Pajek precedentemente reso disponibile per mezzo di una invocazione SCP. La convenzione utilizzata per il nome del file della rete è *idUtenteFB_Timestamp.net*, con ovvio significato delle parti che lo compongono. In maniera opzionale è anche possibile specificare il nome dell'utente Facebook che ha richiesto l'esecuzione per memorizzarlo su database e rendere più immediata l'interpretazione delle tuple.

È innanzitutto essenziale notare come tutto ciò che viene pubblicato sullo stream di Output[1] viene letto e interpretato come risultato dell'esecuzione da chi invoca questa applicazione. Questo permette di evitare di salvare qualsiasi file, evitando così che possano rimanere memorizzati in maniera persistente su Jobe in seguito a eventuali crash di componenti coinvolti nell'esecuzione.

La prima operazione che si rende necessaria in seguito all'invocazione di questa applicazione è la selezione dell'algoritmo. Tale scelta è effettuata per mezzo di una interrogazione al database che restituirà l'elenco degli algoritmi

[1]Viene qui utilizzato lo standard stream messo a disposizione da Java: System.out

attualmente implementati. Da questa lista ne verrà selezionato uno in maniera casuale. Ogni algoritmo è invocato per mezzo di uno specifico comando a cui eventualmente è possibile accodare dei parametri in numero non definito ma con un preciso ordine. Al fine di rendere l'operazione di aggiunta di nuovi algoritmi dinamica, nel database dovranno essere inserite tante tuple quanti sono i parametri e nel preciso ordine in cui questi dovranno essere accodati al comando. Ovvero, l'ordine di accodamento dei parametri al comando avviene in base all'ordinamento crescente degli ID della relativa tupla. Da notare che il nome del file della rete da elaborare sarà un parametro sempre presente e occuperà sempre la prima posizione nella lista dei parametri. Inoltre questo particolare parametro non sarà recuperato dal database in quanto il nome è generato a runtime durante l'esecuzione, poiché come già detto precedentemente segue la convenzione *idUtenteFB_Timestamp.net*.

Ricostruito il comando si può procedere alla sua invocazione tramite la funzione di libreria Java `Runtime.exec()`. Purtroppo tale funzione è soggetta a due problemi: non gestisce la ridirezione di stream e può non terminare. Il primo dei due problemi si verifica in situazioni in cui si cerchi di far eseguire comandi del tipo `java jecho 'Hello World' > test.txt`. Nel caso in cui un tale comando venisse eseguito direttamente in una shell, si otterrebbe come output un file denominato test.txt il cui contenuto è rappresentato dalla stringa 'Hello World'. Eseguendolo invece per mezzo della funzione `exec()` si noterà che il file test.txt non verrà creato e il programma jecho semplicemente scriverà sullo standard output stream "`'Hello World' > test.txt`". Analogo discorso vale anche per comandi in cui la ridirezione si abbia in input, come nel nostro caso accade per l'invocazione di alcuni algoritmi, ad esempio scritti in R [4]. In questi casi, riconoscibili dalla presenza del simbolo "¡" nel comando, occorre rimuovere il simbolo e il nome del file cui si riferisce dalla stringa del comando stesso e gestire programmaticamente la ridirezione. Ovvero, si dovrà recuperare lo stream di input del processo esterno (nell'esempio R) e scriverci da codice il contenuto del file specificato nel comando.

Il secondo problema è più critico. L'invocazione della funzione `exec()` restituisce un oggetto di tipo `Process` sul quale è possibile invocare il metodo `waitFor()` e attendere così l'esecuzione del processo esterno. Può capitare che tale funzione non ritorni mai. Leggendo la documentazione ufficiale di Sun relativa, troviamo quanto segue:

Because some native platforms only provide limited buffer size for standard input and output streams, failure to promptly write the input stream or read the output stream of the subprocess may cause the subprocess to block, and even deadlock.

Dal JavaDoc si ottiene quindi l'indicazione che occorre gestire gli stream, anche se non viene specificato come. La soluzione adottata prevede l'utilizzo di due thread identici ma che lavorano ciascuno su uno dei due stream disponibili, ovvero l'error stream e l'input stream. Il ruolo dei due thread è quello di consumare il contenuto dei buffer non appena questo si rende disponibile, evitando così che si verifichi la situazione illustrata nel JavaDoc. Tale soluzione si è tuttavia mostrata non sufficiente per alcuni processi specifici, quale ad esempio i processi R. Si è quindi optato per un ulteriore raffinamento che sfrutta la sincronizzazione e la conoscenza del formato XML dei risultati. Una volta invocata la funzione `exec()` e avviati i due thread, l'applicazione Cluster si sospende in attesa che il thread che si occupa della gestione del buffer di input lo risvegli una volta ricevuto il tag che indica la fine della ricezione dei risultati. Quest'ultima soluzione si è dimostrata adeguata a risolvere il problema.

Ricevuti i risultati si procede con la loro elaborazione ai fini del salvataggio dei risultati sul database. Così come per i parametri di input, un algoritmo di clustering può avere un numero non precisato di valori di output, oltre ovviamente ai vari cluster. Tali valori sono anch'essi riportati nella stringa XML dei risultati e memorizzati nel database con analoghi discorsi ai parametri di input fatta eccezione del discorso sulle precedenze che in questo caso non si applica.

Per ogni cluster elencato nel risultato, si procede con l'inserimento di una specifica tupla che riporta il numero di utenti che fanno parte di quello specifico cluster. Ovviamente risulterà in principio come non accettata dall'utente. Ogni volta che una tupla è inserita, l'ID che le viene assegnato viene aggiunto come attributo del tag relativo al cluster nella stringa XML dei risultati che verrà restituita all'applicazione web. Questo per avere la possibilità di identificare direttamente la tupla che si dovrà modificare a seguito dell'accettazione della relativa lista da parte dell'utente.

Come ultimo passo rimane quello di restituire i risultati all'applicazione web. Purtroppo anche questa operazione non si è dimostrata banale. All'interno di Jobe il passaggio di stringhe XML tra gli stream dei processi avviene senza nessuna difficoltà. Nel momento in cui però questo passaggio ha come destinazione l'applicazione web (durante lo sviluppo è stato utilizzato Tomcat 6.0.14 [5] come container) tutti i tag XML vengono automaticamente rimossi. Come conseguenza si ha che l'applicazione web vede come risultato una stringa ottenuta come concatenazione del contenuto dei tag, senza i tag stessi, rendendo così impossibile la sua interpretazione. L'unica soluzione funzionante è stata quella di effettuare una conversione da formato XML a

JSON prima della scrittura dei risultati su standard output. Tale operazione è comunque accettabile in quanto i tempi necessari all'esecuzione degli algoritmi di clustering sono tipicamente di molti secondi.

3.3 AccettaLista

Scopo principale di questo modulo è quello di aggiornare i dati di una lista già memorizzata su database, inserendo le informazioni derivate in seguito alla decisione dell'utente di salvare una specifica lista. In figura 3.6 ne è mostrata la struttura.

Nel package `database` sono contenute classi di utilità per gestire l'interazione con il database. Sono resi disponibili metodi per effettuare sia la connessione che le operazioni di manipolazione di base tramite SQL.

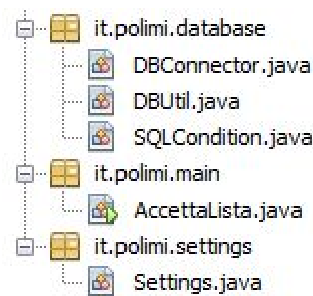

Figura 3.6. I package e le classi dell'applicazione *AccettaLista*.

Il package `settings` contiene la classe adibita a contenitore delle impostazioni dell'applicazione. In particolare sono contenuti eventuali path, dati di accesso e URL dei servizi cui è necessario connettersi.
Il package `main` contiene la classe principale che sarà eseguita in seguito all'invocazione.

Il modulo è reso disponibile su Jobe come file JAR eseguibile ed è invocabile per mezzo del comando `java -jar ./apps/AccettaLista/AccettaLista.jar idLista numAggiunti numRimossi nome`, dove: *idLista* è l'ID della lista per cui è stato deciso il salvataggio e che deve essere coincidente con l'ID della relativa tupla sulla base di dati; *numAggiunti* e *numRimossi* indicano rispettivamente il numero di amici aggiunti e rimossi rispetto a quelli proposti dall'algoritmo di clustering; *nome* è il nome che l'utente ha assegnato alla lista per il suo salvataggio.

La logica applicativa di questo componente risulta essere piuttosto banale, in quanto l'unica operazione da eseguire è una operazione di update sul database.

4 La base di dati

Nel presente capitolo viene illustrata la struttura della base di dati progettata a supporto dell'applicazione e che è mostrata in figura 4.1.

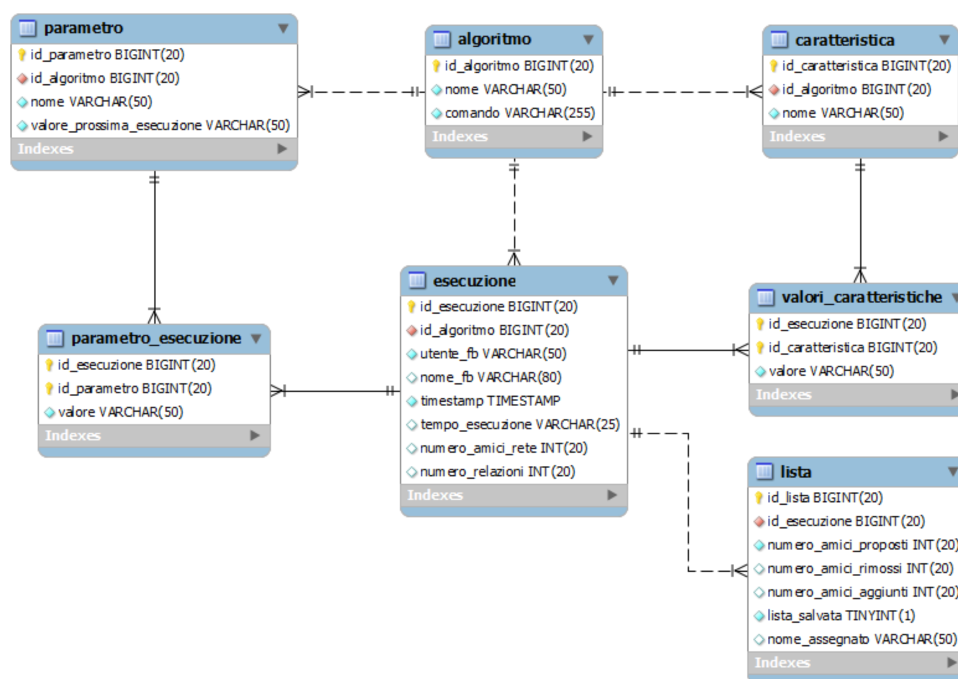

Figura 4.1. Modello ER della base di dati.

Tabella: algoritmo

La tabella contiene tutti gli algoritmi di clustering implementati e resi disponibili su Jobe. Contiene cioè tutti gli algoritmi di clustering invocabili per mezzo dell'applicazione Facebook sviluppata. Utilizzando le informazioni contenute nella tabella *parametro* consente di ricostruire completamente e correttamente il comando da invocare per eseguire uno specifico algoritmo di clustering, così come illustrato nella sezione 3.2.

id_algoritmo La chiave primaria. Viene generata automaticamente in maniera incrementale all'inserimento di una nuova tupla.

nome Il nome dell'algoritmo di clustering. Ad esempio: `R/igraph Betweenness`.

comando Il comando, privo di eventuali parametri, da invocare per eseguire l'algoritmo. Ad esempio: `R < ./algorithms/RIgraphBetweenness-Clustering.r --slave --args`

Tabella: parametro

La tabella contiene la lista di tutti i parametri richiesti per l'invocazione di ogni singolo algoritmo. Per ognuno dei parametri, memorizza il valore che questo assumerà durante l'esecuzione. È importante notare come l'ordine in cui i parametri sono inseriti nel database riflette l'ordine in cui questi devono essere passati da riga di comando al relativo algoritmo. Ad esempio, per un comando del tipo `java -jar algoritmo.jar par1 par2`, la tupla relativa a *par1* dovrà essere inserita per prima nella tabella e di conseguenza avrà ID minore della tuple relativa a *par2*. Infine è supposto che gli algoritmi di clustering abbiano in input sempre almeno un parametro che rappresenta il path del file della rete da analizzare. Tale parametro sarà sempre il primo a dover essere passato e non dovrà essere inserito nel database, in quanto è sempre presente per ogni algoritmo ed inoltre il nome del file è noto solo a runtime e differente per ogni esecuzione[1].

id_parametro La chiave primaria. Viene generata automaticamente in maniera incrementale all'inserimento di una nuova tupla.

id_algorimo Chiave esterna relativa all'attributo *id_algoritmo* della tabella *algoritmo*. Indica l'algoritmo cui il parametro fa riferimento.

nome Un nome che permetta di capire il significato del parametro. Tale informazione non sarà tuttavia sfruttata dall'applicazione.

[1] Ulteriori dettagli sono illustrati nella sezione 3.2.

valore_prossima_esecuzione Il valore che dovrà essere passato all'algoritmo per l'invocazione. La modifica di tale valore influenzerà immediatamente tutte le esecuzioni dell'applicazione successive alla modifica stessa, nonché tutte le esecuzioni già avviate ma che non abbiano ancora selezionato un algoritmo di clustering da eseguire.

Tabella: parametro_esecuzione

La tabella è di utilità per fini statistici e di storico. Poiché il valore di un parametro può essere modificato, tale tabella mantiene traccia del valore che tale parametro aveva per ognuna delle esecuzioni passate.

id_esecuzione Una delle due parti componenti la chiave primaria. E' relativa all'attributo *id_esecuzione* della tabella *esecuzione*.

id_parametro Una delle due parti componenti la chiave primaria. E' relativa all'attributo *id_parametro* della tabella *parametro*.

valore Il valore che un dato parametro aveva durante una passata esecuzione.

Tabella: caratteristica

La tabella, unitamente alla tabella *valori_caratteristiche*, è di utilità per fini statistici. Mantiene traccia degli eventuali valori di output degli algoritmi di clustering. Un esempio di tali valori può essere il numero di archi rimossi dal grafo nel caso di un algoritmo Betweenness Clustering [6].

id_caratteristica La chiave primaria. Viene generata automaticamente in maniera incrementale all'inserimento di una nuova tupla.

id_algorimo Chiave esterna relativa all'attributo *id_algoritmo* della tabella *algoritmo*. Indica l'algoritmo cui la caratteristica fa riferimento.

nome Un nome che permetta di capire il significato della caratteristica. Contrariamente a quanto avviene per l'analogo attributo della tabella *parametro*, tale informazione è utilizzata dall'applicativo. Verranno infatti salvati nella tabella *valori_caratteristiche* solo i valori degli output il cui nome è presente in questa tabella.

Tabella: valori_caratteristiche

La tabella, come descritto precedentemente, mantiene traccia dei valori degli eventuali output di ogni passata esecuzione. Ancora una volta si evidenzia il fatto che i valori memorizzati sono solo quelli degli output il cui nome è stato inserito nella tabella *caratteristica*.

id_esecuzione Una delle due parti componenti la chiave primaria. E' relativa all'attributo *id_esecuzione* della tabella *esecuzione*.

id_caratteristica Una delle due parti componenti la chiave primaria. E' relativa all'attributo *id_caratteristica* della tabella *caratteristica*.

valore Il valore che un dato parametro aveva durante una data passata esecuzione.

Tabella: esecuzione

La tabella principale che permette di risalire a tutti i dati di una singola esecuzione.

id_esecuzione La chiave primaria. Viene generata automaticamente in maniera incrementale all'inserimento di una nuova tupla.

id_algorimo Chiave esterna relativa all'attributo *id_algoritmo* della tabella *algoritmo*. Indica l'algoritmo cui il parametro fa riferimento.

utente_fb L'ID che identifica l'utente su Facebook.

nome_fb Il nome e cognome dell'utente così come registrato su Facebook.

timestamp Timestamp generato automaticamente al momento dell'inserimento della tupla.

tempo_esecuzione Il tempo di esecuzione (espresso in secondi) dell'algoritmo di clustering.

numero_amici_rete Il numero di amici non isolati che compongono la rete di amicizie dell'utente che ha richiesto l'esecuzione, ovvero il numero di amici connessi con almeno un'altra persona che non sia l'utente stesso.

numero_relazioni Il numero di relazioni di amicizia che legano tra loro gli amici di cui sopra.

Tabella: lista

La tabella tiene traccia di tutte le informazioni relative ai cluster/liste ottenuta come risultato delle elaborazioni degli algoritmi di clustering. Inoltre per ognuna memorizza le decisioni che l'utente ha preso in merito.

id_lista La chiave primaria. Viene generata automaticamente in maniera incrementale all'inserimento di una nuova tupla.

id_esecuzione Chiave esterna relativa all'attributo *id_esecuzione* della tabella *esecuzione*. Indica l'esecuzione cui la lista fa riferimento.

numero_amici_proposti Il numero di amici presenti nel cluster così come restituito dall'esecuzione di uno degli algoritmi di clustering.

numero_amici_rimossi Significativo solo se la lista è stata accettata dall'utente, indica il numero di amici che sono stati rimossi dall'elenco proposto prima di effettuare il salvataggio.

numero_amici_aggiunti Significativo solo se la lista è stata accettata dall'utente, indica il numero di amici che sono stati aggiunti all'elenco proposto prima di effettuare il salvataggio.

lista_salvata Indica se la lista è stata o meno accettata dall'utente. Occorre osservare come tale informazione, benché possa sembrare, non è deducibile dai due valori precedenti.

nome_assegnato Significativo solo se la lista è stata accettata dall'utente, memorizza il nome con cui l'utente ha deciso di salvare la lista.

5 Conclusioni e sviluppi futuri

Obiettivo del progetto è stato quello di progettare e realizzare una applicazione Facebook in grado di analizzare la rete di relazioni di amicizia di un utente e di proporre, in seguito all'applicazione di algoritmi di clustering, delle possibili liste di amici. Una lista è intesa come un gruppo di amici legati logicamente da una qualche relazione, quali ad esempio compagni di università o amici di villeggiatura.

Tutto lo sviluppo è stato portato avanti con l'obiettivo di rendere possibile in futuro aggiungere nuovi algoritmi di clustering senza dover modificare il codice dell'applicazione. Inoltre, benché l'applicazione web sia stata sviluppata in linguaggio Java, non si voleva vincolare anche gli algoritmi di clustering a tale linguaggio. L'obiettivo era quello di poter utilizzare anche algoritmi scritti in linguaggi più efficienti. Ulteriore obiettivo del progetto era quello di poter analizzare e confrontare la bontà di diversi algoritmi di clustering. Per raggiungere tale obiettivo è stata progettata e realizzata una base di dati nella quale raccogliere informazioni di ogni singola esecuzione di ciascun algoritmo.

L'applicazione realizzata è al momento sottoposta ad una limitazione imposta da Facebook. Tale limitazione è data dal fatto che non è possibile rendere persistenti sull'account che ha eseguito l'applicazione le liste che sono state proposte e validate dall'utente. Ciò è dovuto al fatto che Facebook non ha attualmente reso disponibili delle API per la gestione delle liste. Tuttavia il problema è stato parzialmente risolto mediante l'implementazione di uno script, esterno all'applicazione, in grado di effettuare comunque tale operazione. Allo script occorre fornire in input un file in formato XML reso disponibile dall'applicazione web.

Sono possibili ulteriori sviluppi del progetto. Considerato che l'attenzione

maggiore è stata data alla possibilità di aggiungere dinamicamente algoritmi di clustering, l'applicazione realizzata si presta in maniera eccellente a essere utilizzata come banco di prova per testare la bontà di tale tipologia di algoritmi. Conseguentemente sarà utile in futuro aumentare il numero di algoritmi implementati. Ancora a proposito di algoritmi, si è notato come alcuni di essi a fronte di reti di amicizie di dimensioni notevoli impieghino un tempo ritenuto eccessivo per un'applicazione web. Può quindi essere utile implementare una funzione di selezione dell'algoritmo di clustering che operi in maniera più intelligente della semplice selezione casuale attualmente implementata. Attraverso tecniche di data mining applicate al database delle esecuzioni possono anche essere trovati quali algoritmi danno migliori risultati a seconda di differenti parametri quali, le dimensioni della rete (numero di nodi e di archi) oppure l'utente stesso che esegue l'applicazione.
Infine è pensabile implementare una funzione che associ dei pesi alle relazioni di amicizia in base a specifici criteri, quale ad esempio il fatto che due amici siano taggati in una stessa foto. Questo genere di informazione può essere sfruttata dagli algoritmi di clustering per proporre cluster ancora più sensati.

Si è tuttavia notato, qualitativamente, come tenendo conto solamente delle semplici relazioni di amicizia tra gli amici dell'utente principale, porti a dei risultati molto soddisfacenti. In particolare ci sembra di poter concludere che l'algoritmo che porta ai risultati migliori tra quelli attualmente implementati[1] è quello di Betweenness Clustering [6]. Per avere risposte più accurate tuttavia sarà necessario studiare l'esecuzione su diverse reti, cosa resa possibile dal database realizzato.

[1] Gli algoritmi implementati al momento della stesura del presente documento sono: Jung Betweenneess Clustering, R/igraph Betweenneess Clustering, R/igraph Walktrap Clustering, Jung Voltage Clustering.

Definizioni dei formati

Come è stato illustrato nei capitoli precedenti, l'esecuzione dell'applicazione sviluppata coinvolge l'invocazione di diversi componenti software. Per far comunicare tra loro tali componenti si è reso necessario definire un formato di scambio comune sfruttando il linguaggio XML. Vengono quindi adesso illustrati i vari formati che sono stati definiti. Data la loro relativa semplicità, per una più immediata comprensione si è preferito riportare degli esempi piuttosto che mostrare la loro definizione mediante DTD.

Risultati del clustering

In figura A.1 è mostrato un esempio di risultati così come restituiti dagli algoritmi di clustering. I dati che è stato necessario riportare sono relativi ai vari cluster individuati comprensivi degli identificativi degli utenti che ne fanno parte. Inoltre, è riportata una lista di eventuali valori rilevanti ai fini informativi identificati dal tag *output*.

Risultati per l'applicazione web

Il formato mostrato in figura A.2 è sostanzialmente lo stesso formato definito per i risultati degli algoritmi di clustering. L'unica differenza è data dal fatto che a ogni cluster viene aggiunto l'identificativo che identifica la tupla relativa nel database. Tale informazione è utile per poter successivamente identificare e modificare tale tupla nel caso in cui l'utente decida di salvare la lista proposta.

```xml
<result>
    <cluster>
        <user id="100002320"/>
        <user id="100073450"/>
        <user id="123434320"/>
    </cluster>
    <cluster>
        <user id="000056550"/>
        <user id="155543456"/>
    </cluster>
    <output name="time">2</output>
    <output name="name">Walktrap</output>
</result>
```

Figura A.1. Esempio del formato definito per i risultati degli algoritmi di clustering.

```xml
<result>
    <cluster id="134">
        <user id="100002320"/>
        <user id="100073450"/>
        <user id="123434320"/>
    </cluster>
    <cluster id="135">
        <user id="000056550"/>
        <user id="155543456"/>
    </cluster>
    <output name="time">2</output>
    <output name="name">Walktrap</output>
</result>
```

Figura A.2. Esempio del formato definito per i risultati da inviare all'applicazione web.

Risultati per l'applicazione web (post JSON)

Come descritto nella sezione 3.2, per inviare i risultati all'applicazione web da Jobe è necessario effettuare una conversione da XML a JSON. Nel momento in cui si esegue la conversione inversa, il formato XML risulta leggermente modificato nella struttura, benché il contenuto informativo sia il medesimo. Ciò è dovuto al fatto che JSON non supporta gli attributi, conseguentemente tutto ciò che in XML era rappresentato come attributo prima della conversione, viene tramutato in un tag figlio al momento della conversione inversa. Tale situazione è illustrata in figura A.3, la quale riporta lo stesso esempio mostrato in figura A.2 dopo il passaggio tramite JSON.

```
<result>
    <cluster>
        <id>134</id>
        <user><id>100002320</id></user>
        <user><id>100073450</id></user>
        <user><id>123434320</id></user>
    </cluster>
    <cluster>
        <id>135</id>
        <user><id>000056550</id></user>
        <user><id>155543456</id></user>
    </cluster>
    <output>
        <name>time</name>
        2
    </output>
    <output>
        <name>name</name>
        Walktrap
    </output>
</result>
```

Figura A.3. Esempio del formato per i risultati come ricevuto dall'applicazione web.

Esportazione

Come illustrato nella sezione 3.1, Facebook non rende disponibili API per salvare le liste di amici. Chi volesse effettuare questa operazione può tuttavia utilizzare lo script già citato, il quale necessita di avere informazioni sulle liste da creare e gli utenti da inserirvi. Dall'applicazione è possibile esportare i risultati come file XML che può essere utilizzato come input dello script. In figura A.4 ne è mostrato un esempio.

```xml
<result>
    <cluster name="PoliMI">
        <user id="100002320"/>
        <user id="100073450"/>
        <user id="123434320"/>
    </cluster>
    <cluster name="Amici mare">
        <user id="000056550"/>
        <user id="155543456"/>
    </cluster>
</result>
```

Figura A.4. Esempio del formato definito per l'esportazione dei risultati.

Acronimi

API Application Programming Interface

DTD Document Type Definition

FBML Facebook Markup Language

HTML HyperText Markup Language

JAR Java ARchive

JSON JavaScript Object Notation

JSP Java Server Pages

SCP Secure CoPy

SNSs Social Network Sites

SSH Secure SHell

SQL Structured Query Language

URL Uniform Resource Locator

XFBML Xml FaceBook Markup Language

XML eXtensible Markup Language

Bibliografia

[1] Pajek wiki. http://pajek.imfm.si/doku.php.

[2] Facebook Developer Wiki. http://wiki.developers.facebook.com/index.php/Main_Page.

[3] jquery autocomplete. http://docs.jquery.com/Plugins/Autocomplete.

[4] The R Project for Statistical Computing. http://www.r-project.org/.

[5] Tomcat. http://tomcat.apache.org/.

[6] Jung Betweennees Clustering. http://jung.sourceforge.net/doc/api/edu/uci/ics/jung/algorithms/cluster/EdgeBetweennessClusterer.html.

[7] When Runtime.exec() won't, 2000. http://www.javaworld.com/javaworld/jw-12-2000/jw-1229-traps.html.

[8] Facebook. http://www.facebook.com/.

[9] Facebook developers wiki. http://wiki.developers.facebook.com/index.php/Main_Page.

[10] Facebook Applications on Glassfish (part 1). http://blogs.sun.com/sduv/entry/write_facebook_applications_using_java.

[11] Writing Facebook Applications Using Java EE. http://www.developer.com/java/article.php/10922_3733751_1/Writing-Facebook-Applications-Using-Java-EE.htm.