JharonSX: Sharing eXperience

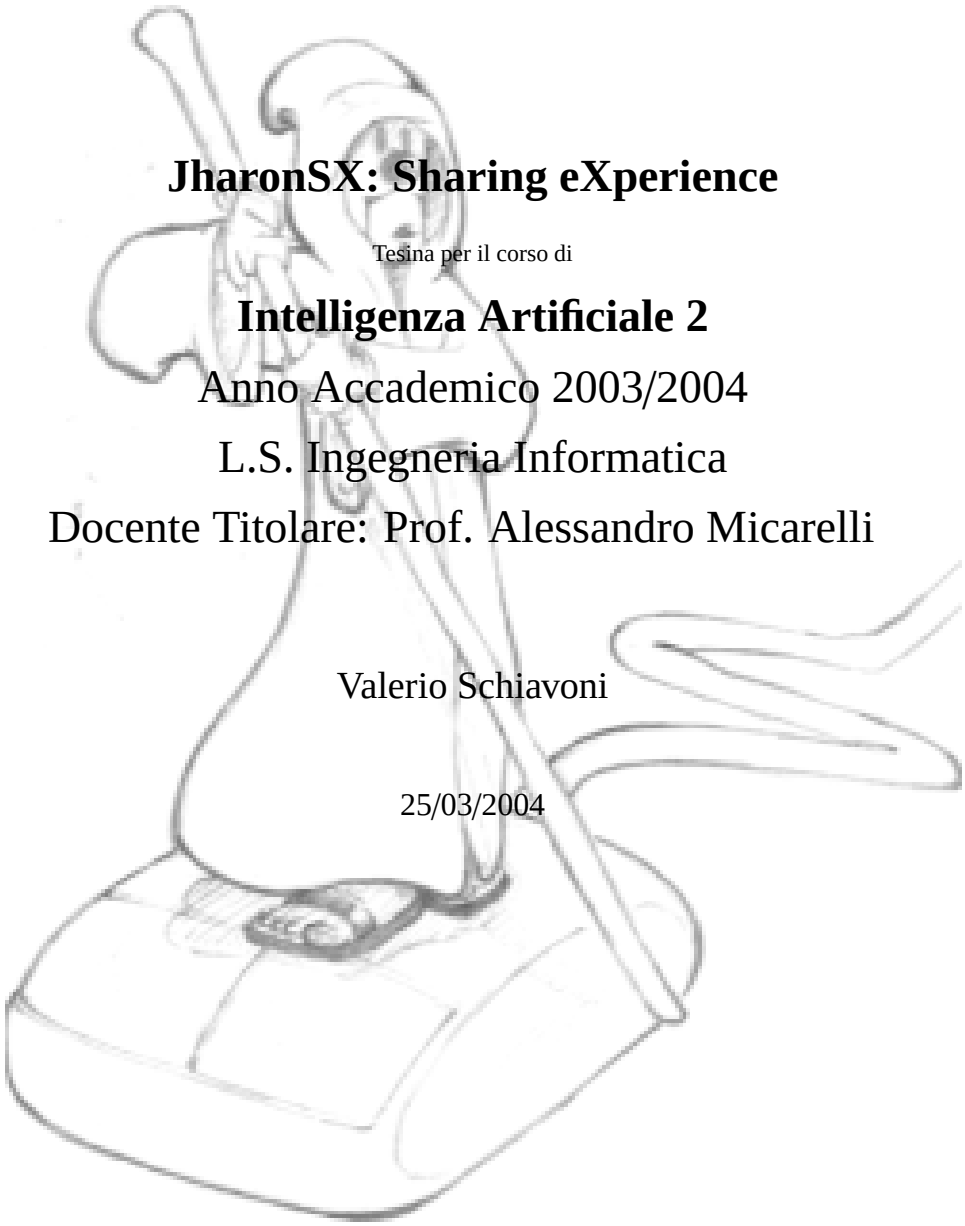
Tesina per il corso di

Intelligenza Artificiale 2

Anno Accademico 2003/2004

L.S. Ingegneria Informatica

Docente Titolare: Prof. Alessandro Micarelli



Valerio Schiavoni

25/03/2004

Indice

Capitolo 1

JharonSX: Sharing eXperience

In questo capitolo verranno illustrate le novità introdotte da JharonSX rispetto al suo predecessore, sia per ciò che riguarda l'architettura software, sia per quanto riguarda i nuovi obiettivi che si vuole perseguire con il suo sviluppo.

1.1 Lavori correlati

Lo studio alla base dello sviluppo di Jharon: *Sharing eXperience* trae ispirazione dall'analisi di altri lavori tesi verso la costruzione incrementale da parte di utenti di sistemi peer-to-peer di database di conoscenza *distribuita*. In particolare per ciò che riguarda la costruzione di un database comune accessibile da tutti gli utenti del sistema si è fatto riferimento a [4], per ciò che la cosiddetta *common knowledge* si è tenuto in forte considerazione [1]; per ciò che riguarda l'adozione di un database embedded installato sulla macchina dell'utente per la gestione dei profili utenti e di un distinto database dalle capacità ben superiori da usare su piattaforma web si è presa ispirazione dall'approccio adottato in [2]; per ciò che riguarda l'utilizzo di un proxy per l'analisi dei header http scambiati tra i browser e il web, in relazione ad un approccio cooperativo nella navigazione si è considerato [3].

1.2 Obiettivi e Motivazioni

Jharon: *Sharing eXperience* nasce dalla volontà di portare avanti il progetto nato durante la tesi di laurea di primo livello dell'autore, e di farlo relativamente allo svolgimento della tesina per il corso di "Intelligenza Artificiale 2" nell'ambito del corso di Laurea Specialistica in Ingegneria Informatica.

Durante lo sviluppo del progetto originale, alcuni aspetti sono stati trascurati, principalmente a causa della mancanza di tempo: da subito era emerso che una delle debolezze di quella prima versione era nell'attribuire un documento ad una delle categorie presenti nel profilo utente attualmente caricato nel sistema, eventualmente creandone una nuova fittizia *foo*, dove molti dei documenti visitati dall'utente finale venivano destinati (questa seconda possibilità si verificava quando non si riusciva a superare un certo threshold relativo all'output calcolato da GAME); tale assunzione di natura semplicistica, vuole essere superata, cercando di realizzare un sistema più furbo.

Per raggiungere tale obiettivo, si useranno delle categorizzazioni che altri utenti hanno fatto contemporaneamente ed autonomamente: in questo modo, nonostante il profilo attualmente usato non riesca ad attribuire un certo documento ad una specifica categoria in maniera puntuale e significativa, si terrà in considerazione, sfruttandola a proprio vantaggio, la categorizzazione effettuata da qualcun'altro nella rete.

Da un punto di vista di sistema cosiddetto *browserware* [1], ci si può chiedere perchè è stata scelta l'architettura, piuttosto che scegliere un'architettura alternativa: nel caso di Jharon le considerazioni fatte in [1] possono essere considerate altrettanto valide, e per tanto non si è intrapresa la strada di un cambio di architettura per l'interazione con l'utente.

Nelle prossime sezioni si illustrerà in dettaglio come tale atteggiamente può essere realizzato, sia da un punto di vista architetturale che implementativo. Rimane un problema aperto verificare quanto l'assegnazione effettuata da un utente sconosciuto possa essere affidabile.

1.3 Architettura Client-Side

L'architettura software di questa nuova versione cambia radicalmente da quella precedente, sia per risolvere delle evidenti carenze, sia per soddisfare le nuove esigenze e per il raggiungimento dei nuovi scopi che il sistema si prefigge. Nelle prossime sezioni verranno illustrate le novità introdotte, in parallelo a quanto è stato riscritto (una percentuale molto alta del software è stata riscritta *from scratch*, da zero).

1.3.1 Diagramma delle classi UML

Il seguente diagramma illustra le classi software relative al modello (si considera come riferimento il pattern MVC).

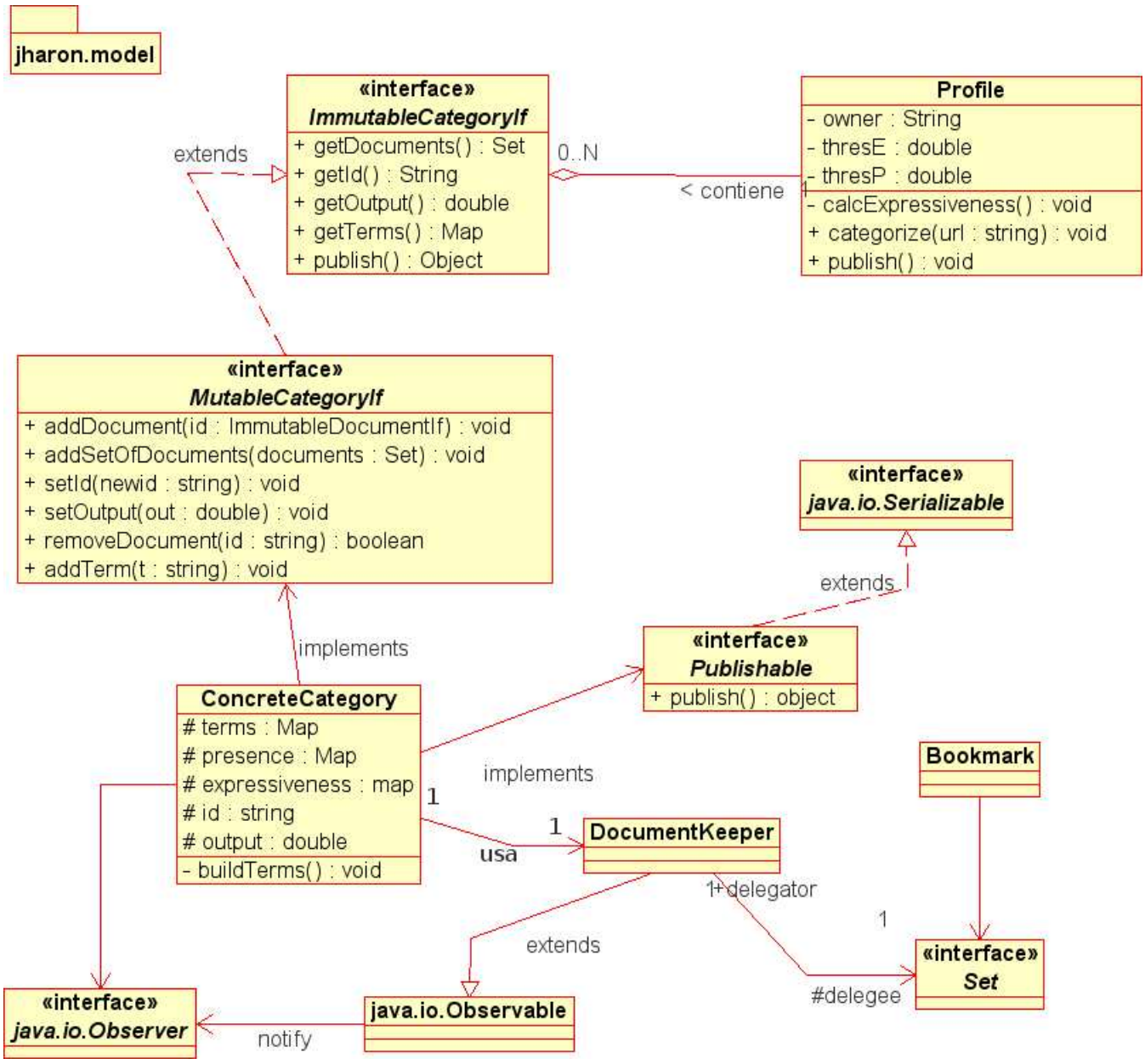


Figura 1.1: Class diagram per il modello

1.3.2 Strato della persistenza, lato client

È stato introdotto uno strato di persistenza dei dati, usato per salvare il profilo utente non più come un semplice file `xml` contenente un albero di categorie e indirizzi di pagine web, ma vengono serializzate per intero le categorie e tutto il loro contenuto. Si è scelto di usare un database embedded come soluzione tecnologica: in particolare è stato usato `SQLite, v2.8.13`, un database engine embeddable, conforme allo standard `SQL92`; questa soluzione mette a disposizione del sistema un ambiente di tipo RDBMS, senza la complessità e l'overhead necessariamente introdotto da soluzioni ben più solide e pesanti (come ad esempio `PostGreSQL`), e senza ovviamente essere un vero RDBMS.

La scelta di questo particolare RDBMS è stata influenza anche dalla disponibilità su web di un wrapper di accesso tramite tecnologia `JDBC`, reperibile all'indirizzo:

`http://www.ch-werner.de/javasqlite/`

Il dump del database è estremamente semplice, essendo costituito da un'unica tabella, ed è illustrato dal listato 1.1.

Listing 1.1: Dump del database SQLite

```
create table categories(
id varchar(30) primary key,
cat blob
)
```

Si può notare come, nonostante il codice appena mostrato dichiari la chiave primaria come tipo `varchar` e l'entità `cat` come tipo `blob`, SQLite è un database typeless. In seguito, istanze della classe `ImmutableCategoryIf` nel package `jharon.model` subiscono una serie di preprocessamenti, per poi essere inserite nel database. La Figura 1.2 illustra i passi che precedono il salvataggio di un profilo utente nel database.

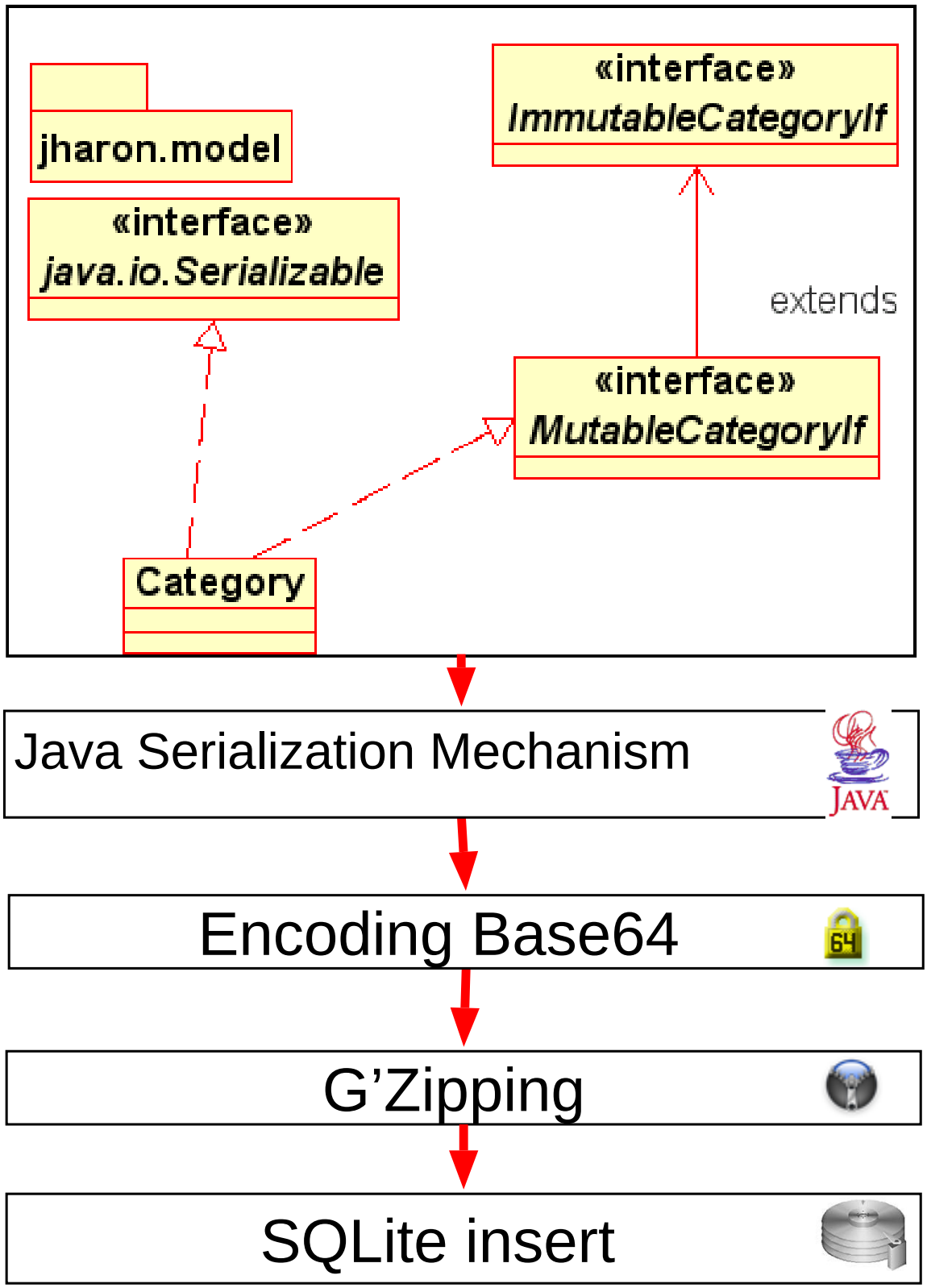


Figura 1.2: Pre-processing e serializzazione

Un'altro motivo che ha portato alla scelta di un database embedded, piuttosto che fermarsi al meccanismo di serializzazione previsto da Java, è quello di poter usare la sintassi e la semantica SQL per poter recuperare le categorie specifiche per un certo profilo, cosa assolutamente non possibile avendo a disposizione degli oggetti software serializzati su files.

Alcune analogie relative all'utilizzo di un database embedded possono essere colte in relazione a Memex [2]. L'architettura di questo sistema, benchè notevolmente più complessa di quella proposta in questa sede, prevede al suo interno l'utilizzo di un Berkeley DB, un altro *lightweight storage manager* usato per mantenere le statistiche relative ai singoli termini: in quella sede viene fatto notare come usare un RDBMS per questi scopi sarebbe, usando un'analogia, come "sparare con un bazooka ad una formica", ed in definitiva comporterebbe uno spreco di risorse in termini di spazio e tempo. In Memex viene inoltre previsto uno strato di sincronizzazione tra due distinti database, per avere un ulteriore grado di coerenza tra ciò che viene salvato nel RDBMS, in questo caso effettivamente presente, e i dati salvati nell database embedded: il team di Memex ha deciso di implementare un sovrastrato di versioning tra i due componenti persistenti; ovviamente tale scelta, oltre ad essere piuttosto complessa, è al di fuori daglo obiettivi del sistema che si propone in questa sede.

1.3.3 Acquisizione di conoscenza

Esistono tre differenti modalità per l'acquisizione di conoscenza da parte del sistema, tutte con il medesimo obiettivo di raffinare la modellazione dell'utente. Esse consistono in:

- a feedback diretto: ereditato dalla precendente versione di Jharon, questa modalità permette all'utente di aggiungere o rimuovere direttamente i documenti alle categorie di appartenenza, definendo in maniera manuale il contenuto del profilo utente; questa modalità delega all'utente molto carico di lavoro, per il quale è stata ideata la seconda modalità, definita auto-adattiva, per le sue peculiarità;
- indiretta tramite proxy (auto-adattivo): introdotta anche questa nella precende versione di Jharon, rappresenta una novità nell'ambito dello user-modeling, permettendo di risolvere, seppur in maniera semplicistica, le evidenti debolezze dell'apprendimento diretto; il suo funzionamento risiete nell'utilizzare gli header dei messaggi http scambiati tra il web browser dell'utente e quelli di risposta dei server nel web, per capire quali siti internet e documenti sono stati visitati, per poterli usare come documenti di training, che vengono automaticamente attribuiti alla categoria stabilita dall'algoritmo GAME usato per categorizzare i documenti; nei casi in cui l'output ottenuto dall'algoritmo risulta sotto un certo threshold (customizzabile), veniva creata una nuova categoria, con label *unknown*, ed in essa veniva aggiunta il documento in esame;
- interrogazione del *knowledge database*: è la *novità* introdotta da questa nuova versione di Jharon, e quella che ha richiesto l'introduzione di una componente server, dove profili di navigazione vengono messi a disposizione da altri utenti del sistema con un'operazione molto semplice da parte degli utenti; l'introduzione di questa componente server permette di acquisire in maniera veloce un profilo utente pre-esistente, per il quale si ritiene che i proprio interessi siano coorenti, ed usarlo per categorizzare i siti internet, oppure i documenti che durante la navigazione ciascun utente incontra;

L'introduzione della componente server ha introdotto quindi la possibilità, per un client del sistema, di poter pubblicare il proprio profilo sul database di conoscenza comune: è proprio in questa possibilità che emerge il significato

dello *sharing experience* associato al nome del sistema. Verranno illustrate nelle prossime sezioni le problematiche di natura tecnologica e teorica che tale possibilità implica.

1.3.4 Interfaccia Utente Lato Client - GUI

In questa sezione verrà illustrata l'interfaccia utente di un client Jharon. L'applicazione è stata scritta usando le librerie `Swing` proprie del `JMF` (Java Multimedia Framework). Rispetto alla versione precedente di Jharon, è stato eliminato completamente quello che veniva chiamato Training Panel, il cui utilizzo permetteva l'addestramento del sistema, potendo aggiungere documenti e categorie al proprio profilo utente. In questa nuova versione, tali funzionalità sono state tradotte in menù a comparsa interattivi interrogabili navigando l'albero che viene mostrato nel pannello `Profile`.

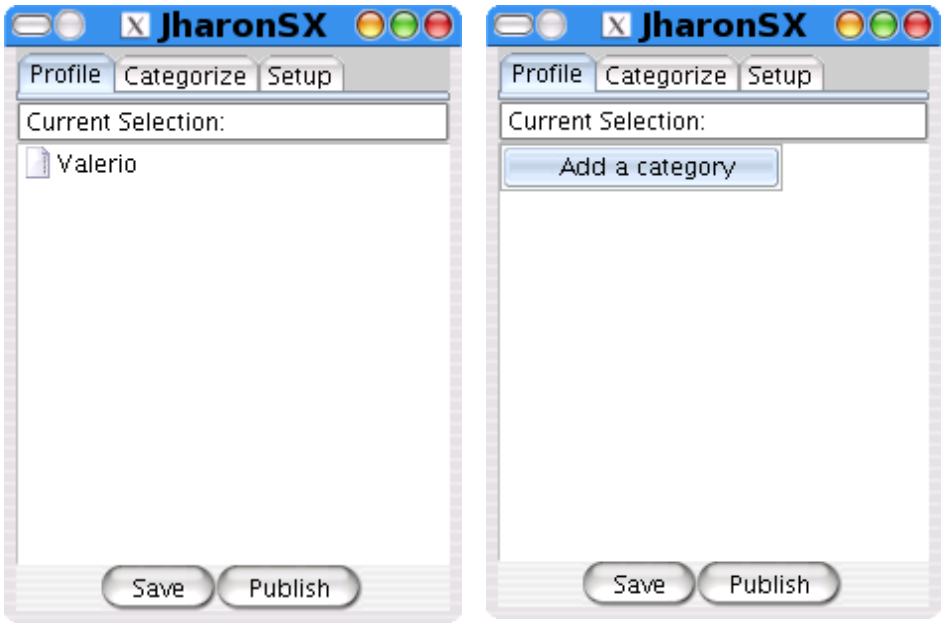


Figura 1.3: Schermata iniziale

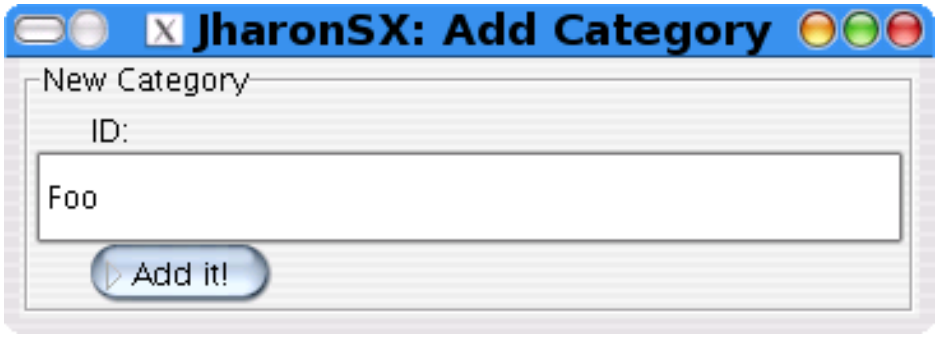


Figura 1.4: Pannello per aggiungere una categoria

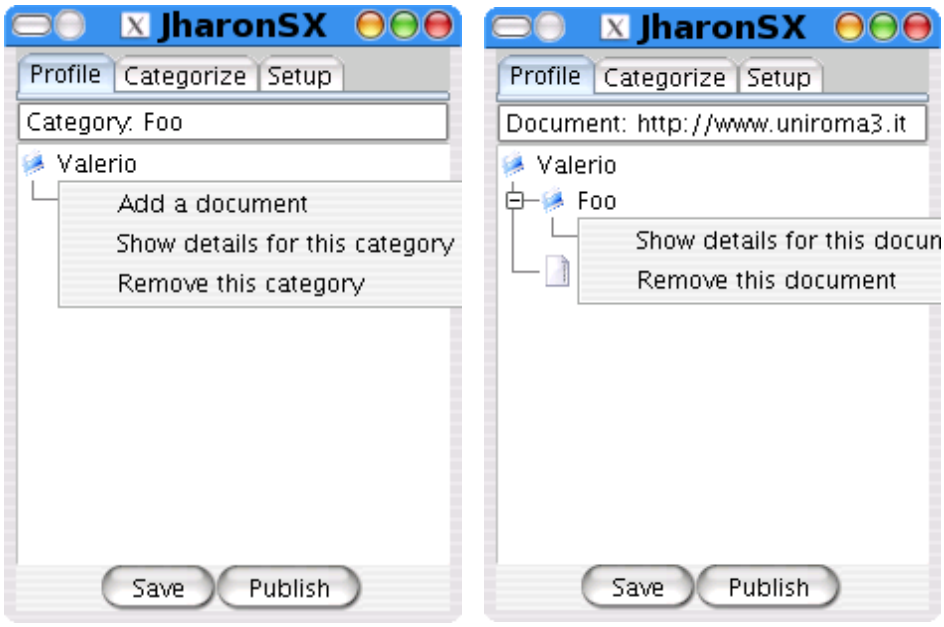


Figura 1.5:

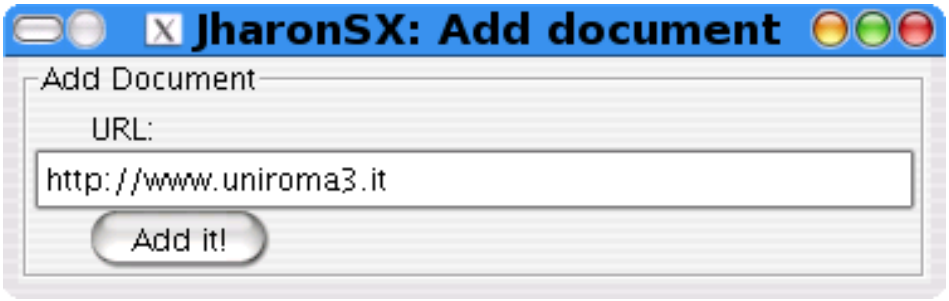


Figura 1.6: Pannello per aggiungere un documento ad una categoria

1.4 Architettura Server-Side

L'architettura software e la tecnologia di supporto ad essa scelte per la componente server di JharonSX sfruttano le possibilità offerte dalla tecnologia Java chiamata RMI; viene fatto inoltre uso delle potenzialità di un RDBMS completo quale è PostgreSQL. Vedremo nel seguito di questa sezione come tali componenti siano state sviluppate, e le funzionalità loro assegnate.

1.4.1 Il server RMI

Analizziamo cosa è RMI e come viene coinvolto in JharonSX. RMI consente di gestire un oggetto remoto, ed è una sorta di client/server trasparente: infatti invece di definire un protocollo e una codifica dei dati per far comunicare client e server, sul server gira un oggetto che viene pilotato dal client, invocando i suoi metodi. Il passaggio dei paramatri e il ritorno di risultati avviene utilizzando la serializzazione, che consente di scrivere e rileggere da uno stream un qualsiasi oggetto Java che supporti la serializzazione. Come si vede nella Figura 1.7, esiste un package `jharon.server` che svolge le funzionalità di RMIServer, capace quindi di accettare chiamate a metodi locali da parte di utenti remoti, quali saranno gli utenti di JharonSX che avranno lanciato un client.

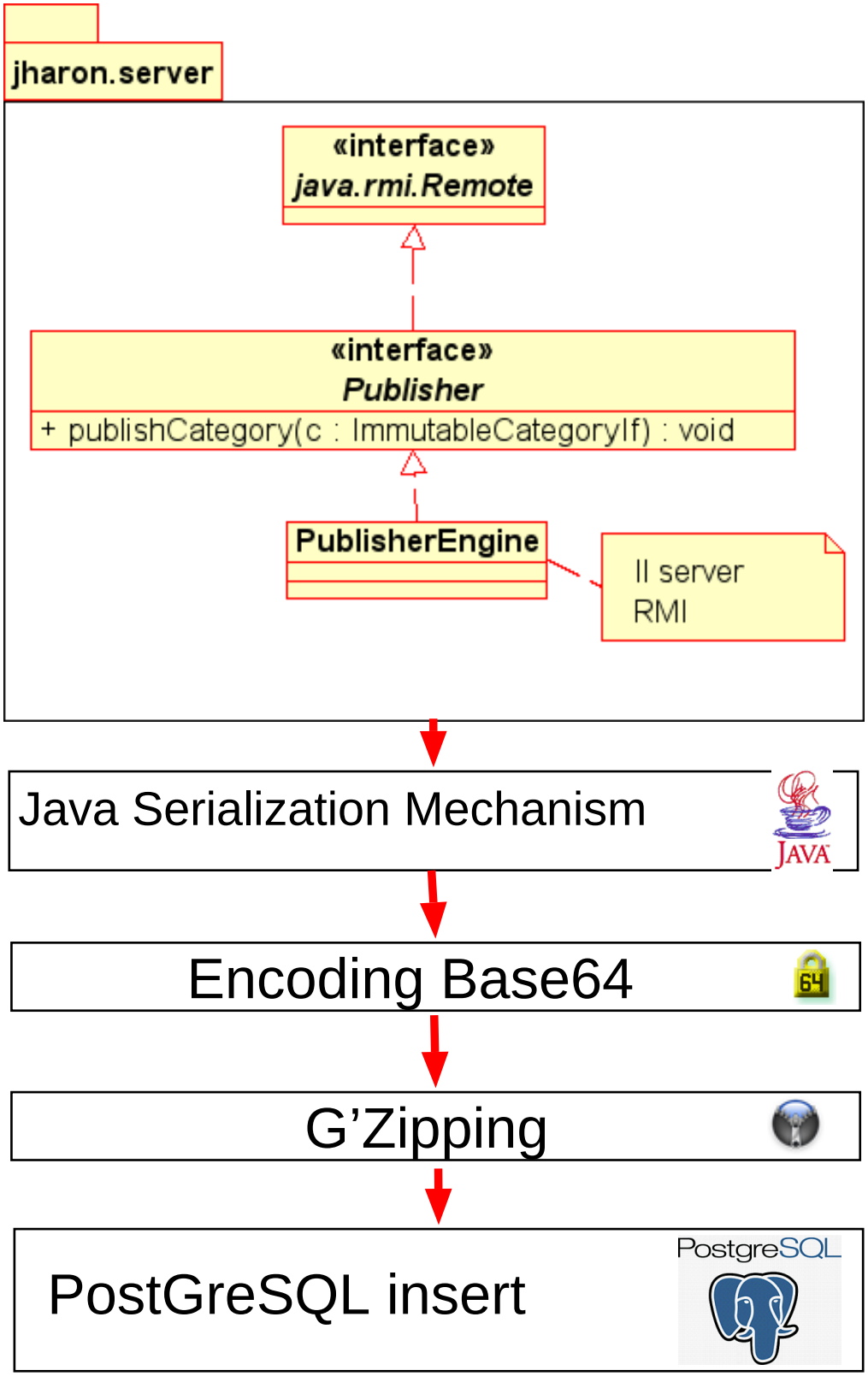


Figura 1.7: RMi Server e Jharon

La funzionalità principale messa a disposizione dal server RMI è quella di permettere la pubblicazione delle categorie di uno specifico profilo dentro un opportuno database PostgreSQL, senza per questo concedere ai client dell'applicazione alcun privilegio in scrittura sul database remoto: sarà il `PublisherEngine` ad occuparsene. Una volta che queste categorie sono state spedite dai client, altri utenti, eventualmente privi momentaneamente di un proprio profilo utente (per i motivi più disparati: il sistema non è stata ancora addestrato, si è conoscenza della presenza di un profilo utente che si addice ai propri interessi, il database locale è stato corrotto e non è più utilizzabile) possono acquisirle; in quest'ottica, il server RMI fornisce, se pur non essendo il suo scopo principale, una funzionalità di *backup* dei profili utenti.

Il codice SQL necessario per la creazione della tabella necessaria a Jharon sul database PostgreSql viene riportato nel listato 1.2

Listing 1.2: Dump del database PostgreSql

```
CREATE TABLE categories (
    idcat character(25) NOT NULL,
    publisher character(30),
    encoded text
);
```

Un'altra funzionalità implementata nello stesso server è quella di permettere l'acquisizione da parte di un client di una certa categoria. In questo modo, un client può arricchire il proprio grado di conoscenza relativamente ad un certo argomento (o categoria) semplicemente chiedendo al server se anche lui conosce la stessa categoria, ed in caso affermativo, aggiungere tra quelli già presenti in una certa categoria locale i documenti attualmente categorizzati come facente parti la stessa, ma presenti solo sul server remoto di conoscenza. Alcune questioni teoriche si pongono subito all'attenzione:

- quanto il server conosce una certa categoria?
- rispetto a cosa un certo documento può essere attribuito dal server ad una certa categoria?
- come può un client sapere se il server conosce esattamente la stessa categoria richiesta dal client?

Il secondo problema è di facile risoluzione: un documento attribuito ad una certa categoria e viene spedito sul server da qualche client, che usando GAME ed il proprio profilo utente, lo aveva attribuito a quella specifica categoria. Da queste banali considerazioni emerge subito come il primo quesito sia strettamente legato al secondo. Per esso però la soluzione non è altrettanto banale, e richiede un ulteriore approfindimento.

Si consideri per esempio il caso in cui un certo documento `a` presente sul server sia stato attribuito e publicato alla categoria `A`. Si immagini inoltre la situazione (non rara) in cui lo stesso documento sia presente nel profilo dell'utente che vuole acquisire come propria la categoria `A`, ma nel suo profilo è presente lo stesso documento `a`, ma localmente al client era stato attribuito alla categoria `B`, in seguito ad un differente modello di addestramento usato dal client. La Tabella 1.1 riassume la situazione descritta. In caso cui nessuna attenzione venga posta al caso di collisioni, il documento `a` potrebbe essere conteporaneamente presente in due categorie. Tale possibilità non può essere ammessa da GAME, per la sua natura di categorizzatore *single-label* [5].

	Client	Server
Documento	a	a
Categoria	B	A

Tabella 1.1: Esempio di collisione

Seguendo lo stesso spirito che ha guidato il concepimento dell'algoritmo GAME, in questa sede di attuerà un approccio *naive*, volutamente semplice. Si consideri la situazione in cui un certo documento è stato assegnato ad una specifica categoria in seguito all'elaborazione compiuta dall'algoritmo in questione, in cui tale operazione avrà prodotto un `output` (per maggiori dettagli, vedere Appendice B.1). Tale output verrà usato come metro di paragone, per risolvere future situazioni di conflitto: esso infatti verrà salvato, insieme con il contenuto di ogni documento relativo ad una specifica categoria, nel database del server; quando un utente verificherà l'incorrere di collisioni, verrà eseguito un semplice algoritmo di *collition avoidance* tra documenti di training: nel caso in cui tale documento sia già presente all'interno del proprio profilo, ma faccia parte di una categoria diversa da quella in cui è stato salvato all'interno del server, verranno confrontati i valori di `output` per i quali quel documento è stato assegnato all'una o all'altra categoria: il documento verrà quindi rimosso da quella in cui tale risultato risulta inferiore.

Incorrere in collisioni può essere considerato anche da un punto di vista diametralmente opposto: un utente cerca di publicare sul server un documento (all'interno della relativa categoria di appartenenza) che però è già stato classificato come facente parte di un'altra categoria: tale operazione potrebbe essere avvenuta in conseguenza di una pubblicazione sul server da parte di un qualsiasi altro utente (ma anche dello stesso che ora lo stà classificando come relativo ad un altro topic); tale possibilità può verificarsi in seguito ad una raffinazione del modello utente, o semplicemente in seguito ad un errore commesso da parte del categorizzatore.

Un problema è rimasto in sospeso: come poter verificare che la richiesta di un client relativamente al contenuto di una certa categoria possa essere risolto in maniera pertinente. Ancora una volta, la natura *naive* di questa architettura emerge, cercando la soluzione più semplice possibile ad un problema apparentemente ben più complesso, cercando poi in lavori futuri, soluzioni più raffinate e furbe. In questa fase comunque, si fa ricorso ad una tecnica spesso usata per la ricerca di documenti o files in ambito file-sharing: viene effettuato l'hash del nome della categoria, usando la funzione di hashing MD5. Nonostante suggerimenti in tale senso vengono forniti per evitare di acquisire conoscenza relativamente ad una specifica categoria, in questo contesto non si fa uso del nome della categoria per derivarne conoscenza relativa al suo contenuto, ma bensì si cerca di evitare collisioni ed di immetere un documento in una errata categoria, andando così a creare la situazione (non voluta) di una categorizzazione *multi-label* [5]. Il controllo che viene quindi effettuato è puramente di natura `sintattica`, e non relativo alla semantica dei loro contenuti. Non avendo trovato in lettura altri esempi di applicazione delle funzioni di hashing per il collision avoidance, si prende per buono l'approccio scelto, salvo approfondimenti e lavori futuri.

1.5 Esempio di sessione di lavoro

Per poter usare il sistema è necessario mandare in esecuzione il server, e poi lanciare un client. Il prototipo del sistema prevede che il server ed il client siano in esecuzione sulla stessa macchina, ma questo non provoca alcuna perdita di generalità: piuttosto, l'uso di prototipi è uno degli strumenti che meglio guidano lo sviluppo e la realizzazione di

sistemi complessi. La distribuzione del sistema prevede questi due archivi `jar` eseguibili: `client-jharon.jar` e `server-jharon.jar`. Si illustrano le procedure necessarie per il test dell'applicazione.

1.5.1 Requisiti software per il server

I requisiti software necessari per lanciare il server sono:

- Java Runtime Environment 1.3.1
- PostgreSql 7.35

Per poter usare il server, è necessario lanciare l'esecuzione dell'`rmiregistry`: a causa di particolari esigenze della tecnologia RMI, è necessario avere salvato nella variabile d'ambiente `CLASSPATH` solo ed esclusivamente l'archivio jar relativo alle classi java per il server. La procedura necessaria viene descritta dal listato 1.3

Listing 1.3: Come lanciare il server

```
hostname:$ unset CLASSPATH
hostname:$ export CLASSPATH="/path/to/server-jharon.jar"
hostname:$ rmiregistry &
hostname:$ java -jar server-jharon.jar
JharonSX is up&waiting for publishers
```

L'ultimo messaggio lasciato dal server conferma che il server è in grado di pubblicare le categorie che vengono postate dai client in esecuzione in ogni parte del mondo.

1.5.2 Requisiti software per il client

I requisiti software necessari per lanciare il server sono:

- Java Runtime Environment 1.3.1
- SQLite

Per lanciare il client, l'unica operazione da compiere è di mandare in esecuzione il programma, specificando come parametro l'IP del server verso il quale si vorranno pubblicare le categorie. In caso di omissione del parametro, verrà assunto come default l'indirizzo di loopback, 127.0.0.1. Quindi per lanciare il client è sufficiente quanto mostrato dal listato 1.4.

Listing 1.4: Come lanciare il client

```
hostname:$ java -jar client-jharon.jar [-IP_server]
```

I prototipi sono stati testati in ambiente Linux e piattaforma hardware x86; inoltre le procedure di deploy dovranno essere perfezionate, attualmente sia i client che il server prevedono un setup dei database corrispondenti prima della rispettiva esecuzione. È comunque possibile incapsulare le istruzioni viste in 1.4 in un unico file eseguibile `bash`. La tipica schermata che l'utente che vede dopo aver appena lanciato il client sulla propria macchina è quella in Figura 1.8. La categoria Linux e i due documenti associati sono appartenenti al profilo di default, comunque modificabile a partire

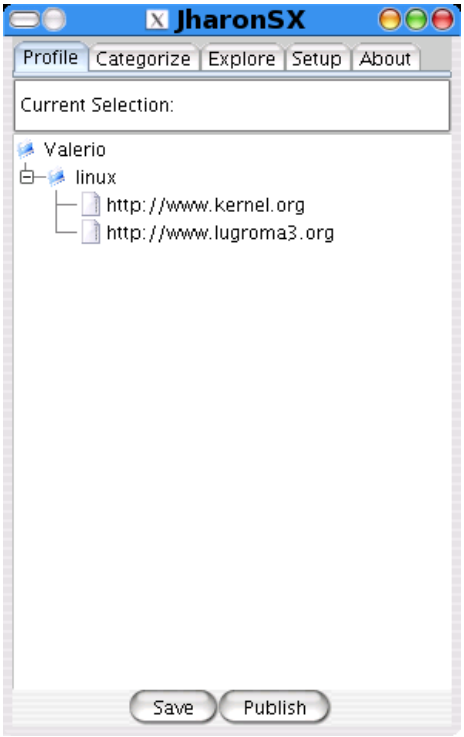


Figura 1.8: Schermata di default, pannello Profile

dal primo avvio dell'applicazione. Nel caso in cui l'utente voglia vedere cosa sia presente sul server remoto, deve accedere al pannello denominato *Explore*. La Figura 1.9 illustra la schermata iniziale in questo pannello. Dopo aver fatto click sul bottone *Explore Jharon Server*, una possibile schermata è quella illustrata in Figura 1.10 Selezionando la riga relativa alla categoria di interesse, il profilo utente viene aggiornato con i dati scaricati dal server. Le Figure 1.11 e 1.12 illustrano quanto detto.

Dopo aver acquisito un profilo utente dal server remoto, oppure averne creato uno manualmente, come illustrato dalle Figure 1.3,1.4, 1.5,1.6, l'utente può provare a categorizare un sito sconosciuto, accedendo al pannello denominato *Categorize*, immettere l'indirizzo di un sito internet, e vedere come GAME assegna tale documento ad una delle categorie presenti nel proprio profilo. La Figura 1.13 illustra il pannello per categorizzare un certo sito internet.

1.5.3 Accesso al codice sorgente

L'intero codice dell'applicazione (sia la componente client che quella server) può essere reperito tramite accesso anonimo pserver sull'account CVS installato su SourceForge.net. Il listato 1.5 illustra la procedura richiesta.

Listing 1.5: Come accedere al codice sorgente

Figura 1.9: Schermata di default, pannello Explore

```
cvs -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/jharon login

    cvs -z3 -d:pserver:anonymous@cvs.sourceforge.net:/cvsroot/jharon co modulename
```

L'intero progetto Jharon:*Sharing eXperience* può essere esplorato visitando il sito:

http://jharon.sf.net/

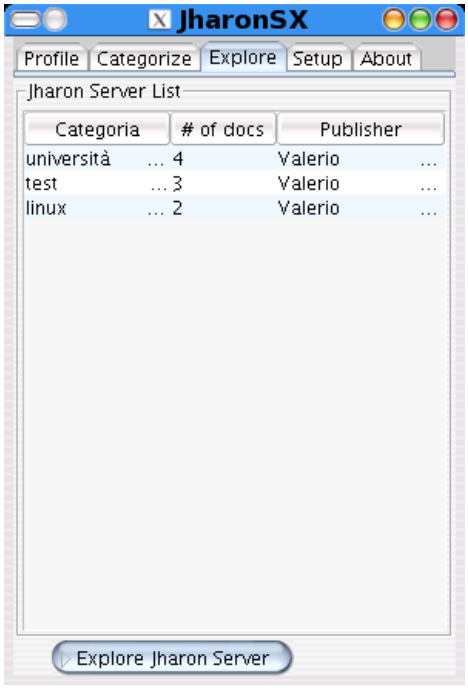


Figura 1.10: Possibile schermata dopo aver esplorato il contenuto del server Jharon

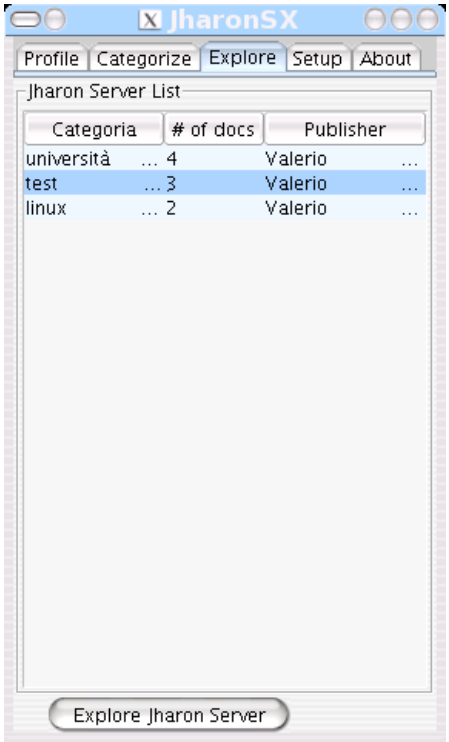


Figura 1.11: Selezione della categoria di interesse

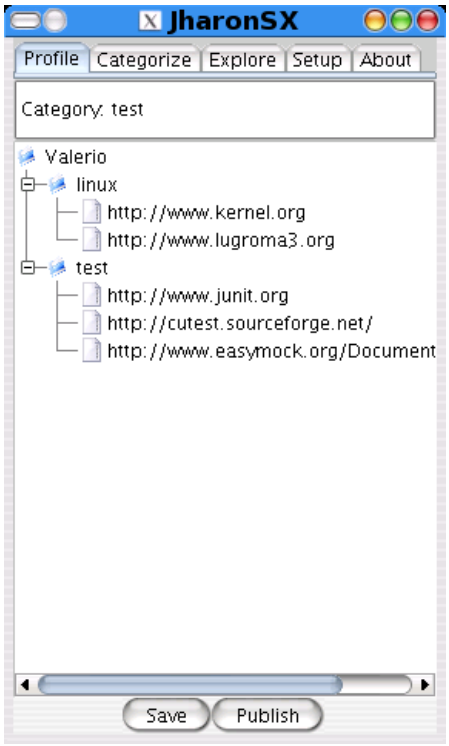


Figura 1.12: Il pannello del Profilo utente, aggiornato con le informazioni prese dal server

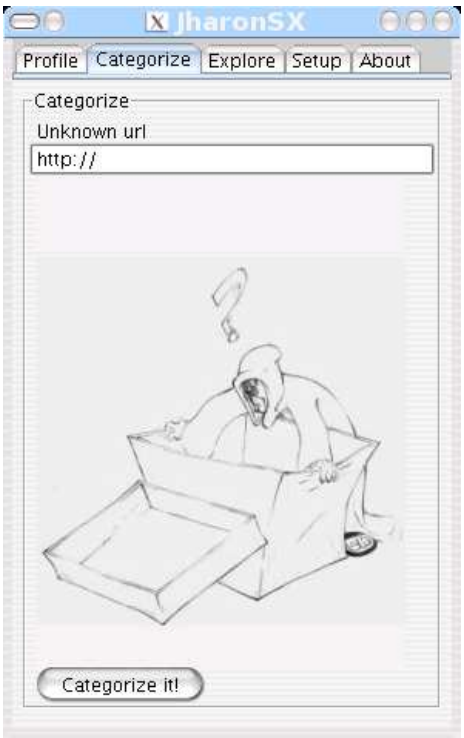


Figura 1.13: Il pannello per categorizzare un certo sito internet

Appendice A

L'algoritmo G.A.M.E.

GAME definisce una metodologia basata sul paradigma Machine Learning studiata per risolvere problemi generali di *pattern recognition*, sviluppata prendendo ispirazione dal RBFNN. In particolare, rispetto al RBFNN, GAME usa *n* funzioni gaussiane mono-dimensionali, una per ogni elemento del vettore di input. Questa modifica permette al sistema di essere addestrato singolarmente rispetto ad ogni dimensione, riducendo la quantità di calcolo richiesta. La maggior parte di questo materiale è stato tradotto da [6].

A.1 Fase di addestramento

Dato un certo dominio di interesse, e definita una particola metrica, durante la fase di addestramento, per ogni classe di oggetti, alcuni esempi di addestramento sono forniti al sistema. Un algoritmo Φ seleziona le *w* parole più rilevanti e calcola per ognuna un vettore ϕ di valori, in accordo con la metrica scelta.

$$\Phi(train\_samples) = \phi = \begin{bmatrix} \{\phi_{1,1}, \phi_{1,2}, \ldots, \phi_{1,j}\} \\ \{\phi_{2,1}, \phi_{2,2}, \ldots, \phi_{2,j}\} \\ \cdots \\ \{\phi_{w,1}, \phi_{w,2}, \ldots, \phi_{w,j}\} \end{bmatrix} \tag{A.1}$$

Φ e la metrica possono variare da obiettivo ad obiettivo.$\phi_{i,j}$ è il j-esimo valore della metrica per la i-esima parola (o *feature*).
Partendo quindi dalla definizione generale di funzione gaussiana:

$$gauss(x) = exp\left(-\frac{(x-\mu)^2}{2 \cdot \sigma^2}\right) \tag{A.2}$$

viene definita, per l'i-esima feature:

$$\mu_i = \omega(\{\phi_{i,1}, \phi_{i,2}, \ldots, \phi_{i,j}\}) \tag{A.3}$$

$$\sigma_i^2 = \psi(\{\phi_{i,1}, \phi_{i,2}, \ldots, \phi_{i,j}\}) \tag{A.4}$$

dove ω e ψ sono due funzioni. Quindi, sostituendo (A.3) e (A.4) in (A.2), viene definita l'i-esima funzione gaussiana come:

$$g_i(x_i, \{\phi_{i,1}, \ldots, \phi_{i,j}\}) = exp\left(-\frac{(x_i - \mu_i)^2}{2 \cdot \sigma_i^2}\right) \tag{A.5}$$

dove x_i è l'i-espimo componente di un nuovo pattern da dover classificare.A questo punto, si può costruisce il vettore caratteristico per la categoria, che è proprio il Gaussian Mono-dimensional Environmente che si vuole trovare. Si può definire quindi:

$$G(x) = \begin{bmatrix} g_1(x_1, \{\phi_{1,1}, \phi_{1,2}, \ldots, \phi_{1,j}\}) \\ g_2(x_2, \{\phi_{2,1}, \phi_{2,2}, \ldots, \phi_{2,j}\}) \\ \ldots \\ g_n(x_n, \{\phi_{n,1}, \phi_{n,2}, \ldots, \phi_{n,j}\}) \end{bmatrix} \tag{A.6}$$

Nella prossima sezione verrà presentato come costruire il vettore x per il nuovo pattern da classificare e come classificarlo.

A.2 Classificazione di un nuovo pattern

Una volta che si è ottenuto il vettore (A.6) delle funzioni gaussiane, è possibile classificare un pattern sconosciuto seguendo la seguente procedura. Per ogni classe di documenti, viene usata la funzione Π per *proiettare* il pattern sconosciuto in un vettore di ingresso x w-dimensionale, dove w è il numero di features della classe.

$$\Pi(pattern) = x = \begin{bmatrix} x_1 \\ x_2 \\ \ldots \\ x_w \end{bmatrix} \tag{A.7}$$

Il vettore x può, o non può, essere usato come metrica nella fase di training, ed è usato come input nell'equazione (A.6).
Quindi, un risultato globale viene per la classe viene calcolato per mezzo della funzione Θ:

$$\Theta(G(x)) = output, \qquad 0 \leq output \leq 1 \tag{A.8}$$

Θ combina l'output di ogni gaussiana, normalizzando il risultato tra 0 e 1. Questa funzione può cambiare da obiettivo ad obiettivo: può pesare in maniera equivalente per ogni g_i, oppure attribuire un peso differente per ognuna di esse. Il pattern viene quindi assegnato alla categoria con il più alto *output* o, eventualmente, a nessuna se l'*output* è inferiore ad un certo *threshold* fissato a priori.

A.3 Caso d'uso: text categorization

In questa sezione viene illustrata l'applicazione di GAME al problema della Text Classification, e la sua applicazione relativa alla classifacazione del testo presente nelle pagine html del web..

A.3.1 Fase di addestramento

La fase di addestramento per un sistema di Text Categorization consiste nel trovare, per ogni categoria, le parole più rilevanti (chiamate appunto *features*) da un insieme di addestramento di documenti, e dare a ciascuna un peso appropriato.

Definizione di Φ Φ calcola le w uniche parole da un set di documenti di addestramento per ogni categoria, e stima due parametri:la Presenza P, e l'Espressività E.P esprime quanto un termine t è presente in documenti appartenenti alla categoria c; E stima quante volte lo stesso termine occorre nei documenti delle altre categorie.
Il vettore ϕ diventa quindi:

$$\phi = \begin{bmatrix} \{E_1, P_1\} \\ \{E_2, P_2\} \\ \dots \\ \{E_w, P_w\} \end{bmatrix} \tag{A.9}$$

Dato un set di categorie $C = \{c_1, c_2, \dots, c_k, \dots, c_n\}$, e dato un numero di documenti m per ogni categoria (ovvero $c_1 = \{d_{1,1}, d_{1,2}, \dots, d_{1,j}, \dots, d_{1,m}, \}$), si possono definire le seguenti quantità:

- Il numero totale di documenti D_{tot} di un dominio è:

$$D_{tot} = \sum_{k=1}^{n} |c_k| = \sum_{k=1}^{n} \sum_{j=1}^{m} d_{k,j} \tag{A.10}$$

- Il numero di documenti D_k della k-esima categoria è:

$$D_k = |c_k| = \sum_{j=1}^{m} d_{k,j} \tag{A.11}$$

- il numero dei document della k-esima categoria nella quale il termine t compare almeno una volta:

$$D_{k|t} = |c_{k|t}| = \sum_{j=1}^{m} d_{k,j|t} \tag{A.12}$$

Quindi, la Presenza P di un termine t nella k-esima categoria è, usando (A.11) e (A.12),:

$$P_{k|t} = \frac{D_{k|t}}{D_k} \tag{A.13}$$

mentre l'Espressività E di un termine t nella k-esima categoria è calcolata usando (A.10), (A.11), (A.12):

$$E_{k|t} = 1 - \frac{\sum_{p=1}^{n} P_{p|t}}{|C| - 1} \qquad p \neq k \tag{A.14}$$

È importante evidenziare che lo stesso termine in differnti categorie ha diversi valorid i espressività.

Definizione di ω e ψ Per l'i-esima feature di una categoria, ω e ψ combinano i paramentri E e P in questo modo:

$$\mu_i = \omega(\{E_i, P_i\}) = E_i \tag{A.15}$$

$$\sigma_i^2 = \psi(\{E_i, P_i\}) = E_i \cdot P_i \tag{A.16}$$

Se poi sostituiamo le due equazioni sopra in (A.2) otteniamo l'i-esima funzione gaussiana, relativa all'i-esima feature:

$$g_i(x_i, \{E_i, P_i\}) = exp\left(-\frac{(x_i - E_i)^2}{2 \cdot E_i \cdot P_i}\right) \tag{A.17}$$

La media della Gaussiana definita in (A.15) può assumere solo valori compresi nell'intervallo (0,1]. Questo vuol dire che la Presenza P di una parola in una categoria può assumere al limite il valore 0 se la parola è rara nella categoria, oppure uguale ad 1 se appare in ogni documento della categoria. La varianza della gaussiana definita in A.16 è sempre compresa nell'intervallo (0,1]. Quando x_i è fissato, il risultato per due differenti funzioni gaussiane può quindi essere comparato.

A.3.2 Fase di categorizzazione

Quando un nuovo documento viene fornito al classificatore, il primo passo è quello di proiettarlo nello spazio di ciascuna categoria, usando la funzione Π. Poi, il risultato di $G(x)$ vine calcolato con Θ.

Definizione di Π Per ogni categoria, se l'i-esima parola della categoria è presente nel documento, allora x_i viene impostato al valore 1, nel caso contrario viene impostato a 0.

$$\Pi(unseen\_document) = \begin{bmatrix} x_1 \\ \dots \\ x_i \\ \dots \\ x_w \end{bmatrix}, \qquad x_i \in \{0, 1\} \tag{A.18}$$

È stato scelto di impostare $x_i = 1$ perchè è la soluzione migliore per normalizzare il massimo risultato della funzione gaussiana ad 1. Infatti, quando la Presenza di un termine t è uguale ad 1 per una particolare categoria (ovvero anche la media della rispettiva g_i è uguale ad 1), allora anche il risultato della stessa g_i è uguale a 1 quando il documento contiene il termine t.

Yahoo! Directory	Acquisiti
Arte	199
Computers	192
Salute	352
Reference	279
Scienza	251
Totali	1273

Tabella A.1: Pagine web dalle directory Yahoo!

Metodo	Performance
FindSim	0.64
Naive Bayes	0.81
BayesNet	0.85
Tree	0.88
GAME	**0.893**
Linear SVM	0.92

Tabella A.2: Misurazione microavaraged F1 sulle 10 categorie più frequenti del Reuters-21578

Definizione di Θ La funzione Θ calcola la media del risultato globale della G(x):

$$\Theta(G(x)) = \frac{\sum_{i=1}^{n} g_i(x_i, \{P_i, E_i\})}{n} \tag{A.19}$$

dove n è il numero di parole nel nuovo documento da classificare. Il risultato di $\Theta(G(x))$ è normalizzato per definizione tra [0,1]. Il documento viene quindi assegnato alla categoria con il più alto valore $\Theta(G(x))$.

Per valutare le performance del sistema nel campo della web categorization, è stata costruita una piccola collezione di documenti HTML dalle categorie di Yahoo!. Il numero totale di documenti acquisiti è riportato nella Tabella A.1

Per ogni pagina web è stato estratto solo il testo, così da poter gestire un numero di parole uniche così da poter comparare i risultati ottenuti con il Reuters-21578.

Risultati Nella Tabella A.2 vediamo quale sia il risultato dei test eseguiti usando GAME ed altri algoritmi allo stato dell'arte. Il metodo FindSim è una variante del metodo di Rocchio, un metodo a feedback per stabilire la rilevanza di un documento. Il classificatore Naive Bayes è costruito usando un insieme di addestramento per stimare la probabilità di appartenza ad una categoria per un certo documento. BayesNet è una rete bayesiana, che usa 2 classificatori bayesiani dipendenti. Tree è un approccio con i Decision Tree.Infine, il Linear SVM definisce un iperpiano lineare che separa l'insime di esempi positivi da quello degli esempi negativi. Come si vede, GAME si piazza appena prima SVM, che è noto essere il migliore approccio machine learning nel campo della classificazione testi. Ciò che è importante sottolineare è che GAME è più veloce di SVM di un ordine di grandezza, e che dovrebbe essere preferito ad esso quando il costo computazionale è di importanza critica. Si dimostra con questi risultati che l'approccio di GAME riesce a risolvere bene questo problema.

Appendice B

Codice Sorgente

B.1 Codice del client

Listing B.1: Main class

```
package jharon;

import java.rmi.*;

import jharon.model.*;
import jharon.persistence.*;
import jharon.gui.*;
import jharon.server.Publisher;

/**Main class, instantiate a Profile, a PersistenceManager and the GUi,
 * and a reference to the remote Publisher Engine.
 *
 * @author Valerio Schiavoni
 * */

public class Client {


    public static void main (String[] args){

        String ipServer ;
        Publisher publ = null;

        System.setSecurityManager(null);
        if (args[0] == null)
            ipServer = "127.0.0.1";
        else
```

```
            ipServer = args[0];

        try{

            String name = "rmi://" + ipServer + "/Publisher";
            publ = (Publisher) Naming.lookup(name);

        }
        catch (Exception e) {
            System.err.println("Client exception: " +
                    e.getMessage());
            e.printStackTrace();
        }
        //pass a reference to the publisher
        Profile profile = new Profile(publ);
        //istantiate a gui and pass a reference to the profile

        Gui gui = new Gui(profile);
        while(true){
            //loop t use gui
        }

    }

}
```

Listing B.2: Gui starter

```
package jharon.gui;

import java.util.*;
import java.awt.*;
import java.awt.event.*;
import java.io.IOException;
import javax.swing.*;

import jharon.model.*;

/**
 *
 * @author Valerio Schiavoni
 *
 */
public class Gui extends JFrame {


    //Specify the look and feel to use.  Valid values:
```

```
    //null (use the default), "Metal", "System", "Motif", "GTK+", "Liquid",
    //"SkinLf"
    final static String LOOKANDFEEL = "Liquid";

    /**
     * 1-parameter constructor
     * A tabbed window is constructed here
     *
     */
    public Gui(Profile p) {

        //using jframe constructor gui sublclasses JFrame
        super("JharonSX");

        initLookAndFeel();

        JFrame.setDefaultLookAndFeelDecorated(true);

        //listener to the close windows event
        this.addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                //                p.save();
                System.exit(0);
            }
        });

        JTabbedPane tabbedPane = new JTabbedPane();

        //add this to the JFrame GUI
        this.getContentPane().add(tabbedPane);
        this.getContentPane().setLayout(new GridLayout(1, 1));

        tabbedPane.addTab("Profile", new TreePanel(p));
        tabbedPane.add("Categorize",new CategorizePanel(p));
        tabbedPane.addTab("Explore", new ExploreServerPanel(p));
        tabbedPane.addTab("Setup", new SettingsPanel(p));
        tabbedPane.addTab("About", new AboutPanel());
        this.setSize(300,300);

        pack();
        show();
        validate();
        setVisible(true);
    }

    /**
     * Inizializzazione per il Look&Feel
```

```
     * TODO: fare in modi che si possa cambiare a run-time
     */
    private void initLookAndFeel() {
        String lookAndFeel = null;

        if (LOOKANDFEEL != null) {
            if (LOOKANDFEEL.equals("Metal")) {
                lookAndFeel = UIManager.getCrossPlatformLookAndFeelClassName();
            } else if (LOOKANDFEEL.equals("System")) {
                lookAndFeel = UIManager.getSystemLookAndFeelClassName();
            } else if (LOOKANDFEEL.equals("Motif")) {
                lookAndFeel = "com.sun.java.swing.plaf.motif.MotifLookAndFeel";
            } else if (LOOKANDFEEL.equals("GTK+")) { //new in 1.4.2
                lookAndFeel = "com.sun.java.swing.plaf.gtk.GTKLookAndFeel";
            } else if (LOOKANDFEEL.equals("Liquid")) { // from external package
                lookAndFeel = "com.birosoft.liquid.LiquidLookAndFeel";
            }
            else {
                System.err.println(
                        "Unexpected value of LOOKANDFEEL specified: "
                        + LOOKANDFEEL);
                lookAndFeel = UIManager.getCrossPlatformLookAndFeelClassName();
            }

            try {
                UIManager.setLookAndFeel(lookAndFeel);
            }
            catch (ClassNotFoundException e) {
                System.err.println(
                        "Couldn't find class for specified look and feel:"
                        + lookAndFeel);
                System.err.println(
                        "Did you include the L&F library in the class path?");
                System.err.println("Using the default look and feel.");
            } catch (UnsupportedLookAndFeelException e) {
                System.err.println(
                        "Can't use the specified look and feel ("
                        + lookAndFeel
                        + ") on this platform.");
                System.err.println("Using the default look and feel.");
            } catch (Exception e) {
                System.err.println(
                        "Couldn't get specified look and feel ("
                        + lookAndFeel
                        + "), for some reason.");
                System.err.println("Using the default look and feel.");
                e.printStackTrace();
```

```
            }
        }
    }

}
```

Listing B.3: Pannello per categorizzare un documento

```
package jharon.gui;

import javax.swing.*;
import java.awt.event.*;
import javax.swing.border.TitledBorder;
import java.awt.*;
import java.awt.image.BufferedImage;
import jharon.model.Profile;
import java.net.URL;
public class CategorizePanel extends JPanel{

    private TitledBorder title;
    private JLabel url = new JLabel("Unknown url");
    private JTextField urlTextField = new JTextField("http://");
    private final JButton btCategorizeUrl = new JButton("Categorize it!");
    private URL imgPath = this.getClass().getResource("/img/categorize.jpg");
    private final Image img = Toolkit.getDefaultToolkit().getImage(imgPath);

    public CategorizePanel(Profile profile){

        Box box = new Box(BoxLayout.Y_AXIS);

        this.title = BorderFactory.createTitledBorder("Categorize");
        btCategorizeUrl.addActionListener(new BtListener(profile));

        box.setBorder(title);
        //box.add(Box.createHorizontalStrut(100));
        box.add(url);
        box.add(urlTextField);
        //box.add(Box.createVerticalStrut(2));
        box.add(new JLabel(new ImageIcon(img)));
        //box.add(Box.createHorizontalStrut(10));
        box.add(btCategorizeUrl);
        this.add(box);
    }

    class BtListener implements ActionListener{
        private Profile p;
        BtListener(Profile p){
```

```
            this.p = p;
        }
        public void actionPerformed(ActionEvent e){
                this.p.categorize(urlTextField.getText());
            }

    }
}
```

Listing B.4: Output

```
/**
 *Assign a new document to its category. Calculate for each category
 *the mean of te activeted Gaussian functions of the terms whose P(i|t)
 *and E(i|t) are greater than ThresP and ThresE.
 *The parameter newDoc is the one called 'test' in the paper.
 *
 *@param url - the url of an unknownd document
 *@return the id if the destination category
 */

public void categorize(String url) {
    //doc to categorize
    HtmlDocument docToCategorize = new HtmlDocument(url);
    double denominatore =
        ((AbstractDocument)docToCategorize).getTokens().size();
    Iterator categoriesIt = categories.iterator();
    //FOR EACH CATEGORY C(i)
    while (categoriesIt.hasNext()) {
        double actualOut = 0; //support variable
        Category cat = (Category) categoriesIt.next();
        //output of the i-th category, resetting every time
        cat.setOutput(0);
        //for each term in presence map of cat...
        Iterator termIter =
            ((Set) cat.getPresence().keySet()).iterator();
        while (termIter.hasNext()) {
            //next term in the presence hashmap
            String term = (String) termIter.next();
            if (docToCategorize.getTokens().contains(term) ) {
                //get presence value for term
                double pVal =
                ((Double) cat.getPresence().get(term)).doubleValue();
                //get expressiveness value for term
                double eVal =
                ((Double) cat.getExpressiveness().get(term)).doubleValue();
                //Check inequality
```

```
                if ((pVal>this.getThresP())&&(eVal>this.getThresE())){
                    //update outputC+ gw(1,t,c)
                    actualOut = cat.getOutput();
                    //t ==eVal , c=pVal ??
                    actualOut = actualOut + Utils.gw(1, eVal, pVal);
                    cat.setOutput(actualOut);
                } //if inequality is true
            } //end if doc containts term
        } //end for each term
        cat.setOutput(actualOut / denominatore);
    } //end for each category
    /*assign the document to the category with the highest output*/
    Category destinationCategory = getHighestOut();
    /*the dictionary of this cat will be updated by the initDictionary
      private method of the category class*/
    destinationCategory.addDocument(docToCategorize);
    update();
} //END CATEGORIZATOR
```

Listing B.5: Codice per il PersistenceManager lato client

```
package jharon.persistence;

import jharon.model.*;
import java.io.*;
import jharon.utilities.Base64;
import java.util.Set;
import java.util.HashSet;
import java.sql.*;
import SQLite.*;
/**
 * A persistence manager, to read and save a user profile from/to a db.
 * A user profile is taken from local db; a community-profile
 * has to be taken from a remote db.
 *
 * @author Valerio Schiavoni
 */
public class PersistenceManager {
    private final static String driver = "SQLite.JDBCDriver";
    /*name of the file where all persistence side is saved*/
    private final String dbname = "jdbc:sqlite:/jharon.db";
    private Connection connection = null;
    /**
     * Default constructor
     * @param dbname the name of the db ;
     *
     */
```

```
    public PersistenceManager(){
        try{
            Class.forName(this.driver);
            connection = DriverManager.getConnection(this.dbname);
        }
        catch(SQLException e){
            e.printStackTrace();
        }
        catch (ClassNotFoundException e){
            e.printStackTrace();
        }
    }
    /**
     * Remove a category from the db.
     * @param id the the id of the category to remove
     */
    public void removeCategory(String id){
        Statement stm = null;
        String sqlRemove = "delete from CATEGORIES where id='"+id+"'";
         try{
            stm = connection.createStatement
                (ResultSet.TYPE_SCROLL_INSENSITIVE,
                 ResultSet.CONCUR_READ_ONLY);
            stm.executeUpdate(sqlRemove);
        }
        catch (SQLException e){
            System.err.println("PersistenceManager.java, serializeCategory");
            e.printStackTrace();
        }
    }
    /**
     * Serialize a category into the db.
     * The object is first encoded in a base-64 string, insert into the
     * db as a string column, where the key is the id of the category
     *
     * @param c a category to be serialized;
     * @return void
     */
    public void serializeCategory(ImmutableCategoryIf c){
        Statement stm = null;
        /*encodeObject need a Serializable-implementing object*/
        String encCategory =
            Base64.encodeObject
                ((Category)c, Base64.GZIP | Base64.DONT_BREAK_LINES);

        String sqlInsert =
        "insert into CATEGORIES values('"+c.getId()+"','"+encCategory+"')";
```

```
        String sqlRemove =
        "delete from CATEGORIES where id='"+c.getId()+"'";
        try{
            stm = connection.createStatement
                (ResultSet.TYPE_SCROLL_INSENSITIVE,
                 ResultSet.CONCUR_READ_ONLY);
            /*remove old occurrence*/
            stm.executeUpdate(sqlRemove);
            /*insert new occurence*/
            stm.executeUpdate(sqlInsert);
        }
        catch (SQLException e){
            System.err.println("PersistenceManager.java, serializeCategory");
            e.printStackTrace();
        }
    }
    /**
     * Deserialize a category.
     * @param id of a given category
     * @return a istance of a MutableCategoryIf
     *
     */
    public MutableCategoryIf deserializeCategory(String id){
        MutableCategoryIf serializedCategory = null; //to be returned
        /*insert the category into the db*/
        String sqlSelect = "select cat from CATEGORIES where id='"+id+"'";

        Statement stm = null;
        try{
            stm = connection.createStatement
                (ResultSet.TYPE_SCROLL_INSENSITIVE,
                 ResultSet.CONCUR_READ_ONLY);
            ResultSet res = stm.executeQuery(sqlSelect);
            while (res.next()){
                /*encoded category is in the first column of the result*/
                String encCategory =
                    (String) res.getString(1);
                serializedCategory =
                    (MutableCategoryIf) Base64.decodeToObject( encCategory);
            }
        }
        catch (SQLException e){
            e.printStackTrace();
        }
        return serializedCategory;
    }
    /**
```

```
     * Delete all from db's table called CATEGORIES
     */
    public void deleteAllFrom(String table){

        StringBuffer sqlSelect = new StringBuffer("delete from ");
        sqlSelect = sqlSelect.append(table);
        Statement stm = null;
        try{
            stm = connection.createStatement
                (ResultSet.TYPE_SCROLL_INSENSITIVE,
                 ResultSet.CONCUR_READ_ONLY);
            stm.executeQuery(sqlSelect.toString());
        }
        catch (SQLException e){
            e.printStackTrace();
        }
    }
    /**
     * Deserialize all the categories stored in the embedded DB.
     *
     * @return a set of jharon.model.Category
     * @see jharon.model.ImmutableCategoryIf
     */
    public Set deserializeCategories(){
        Set categories = new HashSet(); //to be returned
        /*insert the category into the db*/
        String sqlSelect = "select * from CATEGORIES ";

        Statement stm = null;
        try{
            stm = connection.createStatement
                (ResultSet.TYPE_SCROLL_INSENSITIVE,
                 ResultSet.CONCUR_READ_ONLY);
            ResultSet res = stm.executeQuery(sqlSelect);
            while (res.next()){

                String idCat = (String) res.getString(1);
                String encCategory = (String) res.getString(2);
                MutableCategoryIf serializedCategory =
                    (MutableCategoryIf) Base64.decodeToObject( encCategory );
                categories.add(serializedCategory);
            }
        }
        catch (SQLException e){
            e.printStackTrace();
        }
        return categories;
```

```
        }
}
```

Bibliografia

[1] H. Marais, K. Bahrat: *Supporting Cooperative and Personal Surfing with a Desktop Assistant*, ACM Symposium on User Interface Software and Technology, pages 129-138, 1997

[2] Chakrabarti, Srivastava, Subramanymam, Tiwari: *Using Memex to archive and mine community Web browsing experience*, Computer Networks 33(1-6): 669-684 (2000)

[3] Cabri, Lenoardi, Zambonelli: *Supporting Cooperative WWW Browsing: a Proxy-based Approach*, 7th Euromicro Workshop on Parallel and Distributed Processing, Madeira, Portugal, 1999, 138-145.

[4] Gnasa, Alda, Grigull, Cremers: *Towards Virtual Knowledge Communities in Peer-to-Peer Networks*, Distributed Multimedia Information Retrieval 2003: 143-155

[5] Sebastiani: *Machine Learning in Automated Text Categorization*, ACM Computing Surveys 34(1), 2002.

[6] DiNunzio Micarelli.*Does a new simple gaussian weightning approach perform well in text categorization?* Proc. of the Eighteenth International Joint Conference on Arti cial Intelligence ,IJCAI-03, Acapulco, Mexico, 9-15 August 2003